A new computational framework for speeding up the fitting of multistate capture–mark–recapture models

Matia H. Muller[1*], Jaume A. Badia-Boher[1§]

[1]Population Biology Research Unit, Swiss Ornithological Institute, Seerose 1, 6204 Sempach, Switzerland

* Corresponding author: matia.muller@vogelwarte.ch

§ Senior author

Running headline: Speeding up multistate mark-recapture models

Acknowledgements

We thank Marc Kéry and Michael Schaub for reviewing the manuscript and providing valuable advice. We thank Christof Herrmann and the Hiddensee ringing centre for coordinating and providing the ringing and resighting data, and all ringers, observers and collaborators who contributed to data collection.

Data availability statement

Data and code are available for peer reviewers only and will be permanently available upon acceptance in a journal. Anybody interested in the data and code from the preprint can send an email to any of the authors.

Conflict of interest statement

The authors declare no conflict of interest.

Author contributions

Matia H. Muller and Jaume A. Badia-Boher conceived the main ideas of the study. Matia H. Muller developed the method for the case study. Both authors performed the analyses and wrote the manuscript equally.

Abstract

1. Multistate capture–mark–recapture (CMR) models are widely used to estimate the parameters governing demographic processes such as survival, dispersal, and recruitment in animal populations. In Bayesian analyses, the multinomial likelihood of multistate CMR models summarizes individual encounter histories into groups defined by states and capture occasions, and is regarded as a computationally efficient alternative to widely-used state-space models. However, as model complexity increases – through additional states or capture occasions – the conventional multinomial formulation can also reach very long or intractable computation times. Depending on model structure, the multinomial formulation may include many unnecessary calculations that can be removed while preserving an equivalent model, potentially resulting in substantial computational gains.
2. Here, we present an approach to eliminate such unnecessary calculations, and therefore optimize the implementation of multistate multinomial CMR models. First, we describe the modifications needed for optimization under commonly used multistate model variants: movement models where states represent discrete sites,

and age-structured models where states represent age classes. We also provide code-based tutorials to facilitate method implementation. To evaluate model performance, we simulated CMR data under both model types with increasing levels of complexity (two to four sites, two to four age classes), fitted conventional and optimized formulations in NIMBLE and JAGS, and compared parameter estimates in terms of bias, coverage, and computational efficiency. In a case study, we also fitted both formulations to a dataset of German White Storks, using a model that combines age structure and movement between regions.

3. The optimized formulation produced equivalent estimates to the conventional formulation across simulations and in the case study, confirming that both formulations implement the same likelihood. Relative gains in computational efficiency increased with model complexity in age-structured models, reaching a factor of 13 in NIMBLE and 18 in JAGS. Gains were more modest in movement models, which was expected as their structure is less optimizable. In the case study, the optimized formulation was 11-fold more efficient in NIMBLE and 5-fold in JAGS.
4. Our results illustrate that Bayesian multistate CMR models can be largely speeded up, permitting the analysis of more complex models and larger datasets.

Keywords

Capture-mark-recapture – Multistate models – Computational efficiency – Bayesian – JAGS – NIMBLE – Multinomial likelihood

1. Introduction

Capture-mark-recapture (CMR) models are a widely used class of statistical models in ecology to analyze the encounter histories of marked individuals to infer demographic processes in animal populations (Cooch & White, 2019). Early open-population CMR models, like the Cormack-Jolly-Seber model, focused on the estimation of survival and detection probabilities (Lebreton et al., 1992). The development of multistate models extended this framework by allowing individuals to transition among discrete biological or spatial states (e.g., “breeder” or “non-breeder”, “infected” or “healthy”, “in site 1” or “in site 2”), with demographic parameters such as survival and dispersal, as well as detection probabilities, varying according to the state of the individual (Nichols et al., 1992, Lebreton et al., 2009). This innovation greatly expanded the scope of open-population CMR models, enabling inference on processes of central importance in ecology, conservation, and epidemiology, such as dispersal, recruitment, or disease dynamics (Conn & Cooch, 2009; Schaub & Kéry, 2022). As a result, multistate CMR models have become a cornerstone of modern demographic analyses across a wide range of ecological systems.

The development of Bayesian hierarchical implementations of CMR models, facilitated by general-purpose probabilistic programming languages such as BUGS and its derivatives, has allowed one further step in terms of modelling flexibility, uncertainty quantification and uncertainty propagation in multistate models (Plummer et al., 2003; de Valpine et al, 2017, Schaub & Kéry, 2022). Another considerable benefit of Bayesian implementations is that it

eases integration of CMR data with other datasets, for example in Integrated Population Models (IPMs; Besbeas et al., 2002; Schaub & Abadi, 2011). This degree of flexibility and integration capacity has not been matched by any maximum-likelihood (ML) software or interface.

The flexibility of Bayesian implementations, however, comes at a substantial computational cost. Fitting Bayesian multistate CMR models in the BUGS language takes substantially more time than with ML alternatives (Labuzzetta et al., 2024). Such longer runtimes can become a limitation with increasing CMR dataset size (number of individuals, number of capture occasions), model complexity (e.g., number of states or parameters, use of widely implemented temporal random effects, or models that incorporate a spatial dimension), or when CMR models are embedded into integrated frameworks. Excessive computation times can limit the ability to fit complex models, restrict the exploration of different modelling structures (model selection), hinder the application of models in applied ecological settings, and ultimately slow progress in ecological research. These limitations highlight the need for computational strategies that make Bayesian multistate CMR models faster to fit (Yackulic et al., 2020).

Several formulations exist for analyzing multistate CMR data. In the Bayesian framework, the multinomial likelihood is widely regarded as a computationally efficient alternative to the widely used state-space likelihood (Schaub & Kéry, 2022). Its computational efficiency arises from aggregating individual capture histories into so-called m-arrays, which summarize, for each release occasion and state (i.e., “cohort”), the numbers of individuals re-encountered across later occasions and states. The cell counts associated with each

release cohort can then be modelled using a multinomial distribution. Even if the multinomial likelihood often provides substantial computational gains, it relies on the strategy of formulating additional states to accommodate discrete individual covariates that are very often used in multistate models, like different age classes or sites. As the number of states and sampling occasions increase, m-arrays grow rapidly in dimension and in the number of elements they contain, leading to a concomitant increase in computational costs. Consequently, despite its reputation for efficiency, the conventional implementation of the multinomial model can become computationally demanding or even intractable as model complexity increases (Schaub & Kéry, 2022).

Here, we present an approach to improve the computational efficiency of multistate CMR models in the multinomial likelihood. The approach is based on simplifying the structure of the m-array while retaining the same likelihood as the full m-array. We identify and remove redundant or structurally impossible transitions that contribute no information to parameter estimation, therefore reducing unnecessary calculations. We illustrate the logic of this optimized implementation using two widely applied classes of multistate models as examples: (i) *movement models* in which transitions among discrete locations are represented as state transitions, and (ii) *age-structured models* with dead recoveries, in which age classes are treated as states. For each model class, we first describe the modifications needed to optimize the implementation of the model; and second, we fit the model to simulated datasets and compare the resulting parameter estimates and computational efficiency to those of models fitted in the conventional multinomial framework. Furthermore, to assess how the performance of the optimized implementation

scales with model complexity, we examine models with increasing numbers of states in both examples. All analyses are conducted using two widely used Bayesian BUGS-based inference engines for CMR analyses: NIMBLE (de Valpine et al., 2017) and JAGS (Plummer et al., 2003). Finally, we apply the optimized multinomial implementation of the model to a case study with German White storks (*Ciconia ciconia*). For this case, we develop a more complex model that combines age classes and movement among discrete sites. We expect the optimized multinomial implementation to contribute to the development of faster, more complex models that contribute to advances in ecology, conservation, demography, and epidemiological sciences.

2. Methods

2.1 Conventional implementation of the multinomial model

In the conventional multinomial formulation, data are arranged as an m-array (Burnham, 1987), which summarizes individual-based mark–recapture histories into a matrix. Each row of the m-array corresponds to a release cohort, defined by the release state of a marked individual and the time of its release. Let S denote the number of states and T the number of capture occasions. Releases from the last capture occasion are not included in the m-array, as these individuals cannot be subsequently re-encountered and thus contain no information about the parameters. The m-array contains thus $S(T-1)$ rows in total, one for each state-time combination (s,t) with $s \in \{1,\dots,S\}$ and $t \in \{1,\dots,T-1\}$. Each column corresponds to the first re-encounter of these individuals in a specific state–time

combination, and there is one additional column for individuals never re-encountered. Re-encounter occasions range from 2 to T, so that the first column ($k = 1$) corresponds to individuals re-encountered at time 2. Hence, there are $S\,(T-1)\,+\,1$ columns in total. For simplicity, both rows (release cohorts) and columns (re-encounter categories) are numbered sequentially: $c = 1\,, \ldots, S(\,T-1\,)$ for release cohorts and $k = 1\,, \ldots, S\,(T-1)+1$ for re-encounter categories, even though the time indices underlying them start at $t = 1$ and $t = 2$, respectively. After every re-encounter, an individual is considered newly released into the cohort defined by its current state and time. Figure 1 shows in details how an m-array is obtained from individual capture-histories.

capture-histories

	O1	O2	O3	O4
Individual 1	0	2	2	1
Individual 2	1	2	2	0

0 = not seen
1 = seen in state 1
2 = seen in state 2

m-array

	O2S1	O2S2	O3S1	O3S2	O4S1	O4S2	Never
O1S1	0	**1**	0	0	0	0	0
O1S2	0	0	0	0	0	0	0
O2S1	0	0	0	0	0	0	0
O2S2	0	0	0	**2**	0	0	0
O3S1	0	0	0	0	0	0	0
O3S2	0	0	0	0	**1**	0	**1**

FIGURE 1. Simple example illustrating how individual capture histories are aggregated into an m-array (2 individuals, 4 capture occasions, 2 states). The cell (O1S1; O2S2) contains a value of 1 because one individual was first captured at occasion 1 in state 1 and first re-

encountered at occasion 2 in state 2. The cell (O2S2; O3S2) contains a value of 2 because both individuals were captured at occasion 2 in state 2 and first re-encountered at occasion 3 in state 2. The cell (O3S2; O4S1) contains a value of 1 because one individual was captured at occasion 3 in state 2 and first re-encountered at occasion 4 in state 1. Finally, the cell (O3S2; Never) contains a value of 1 because one individual was captured at occasion 3 in state 2 and never re-encountered. All other cells contain zeros because the corresponding transitions were not observed for the two individuals.

For each release cohort c, the corresponding row $\boldsymbol{m}_c$ of the m-array is assumed to follow a multinomial distribution,

$$\boldsymbol{m}_c \sim Multinomial\ (R_c, \boldsymbol{\pi}_c), \qquad \text{(eqn. 1)}$$

where R_c is the total number of individuals released in cohort c and $\boldsymbol{\pi}_c = (\pi_{c,1}, \pi_{c,2}, \dots, \pi_{c,K}, \pi_{c,\emptyset})$ is the vector of probabilities of first re-encounter in each possible time-state combination $k = 1, \dots, K$, plus the probability of never being re-encountered $\pi_{c,\emptyset}$. By definition, $\boldsymbol{\pi}_c$ sums to 1 within a release cohort.

In the conventional multinomial formulation, the probabilities $\pi_{c,k}$ are derived from two matrices (Kéry & Schaub, 2012). The first is the state-transition matrix Ω_t, an $S \times S$ matrix (S being the total number of states), whose element (s, r) gives the probability of transitioning from state s at time t to state r at time $t + 1$. The second is the $S \times S$ observation matrix Θ_t, which describes the probability of being re-encountered in state s at time t if in state r at time t. To obtain the cell probabilities $\pi_{c,k}$, these matrices are

repeatedly multiplied across all combinations of release cohorts and subsequent re-encounter occasions. Details about the computation of $\pi_{c,k}$ probabilities are given in Appendix S1.

In total, $2(T-1)^2-1$ matrix operations are needed, each involving $2S^3-S^2$ arithmetic operations. Due to the way BUGS language packages like JAGS and NIMBLE perform matrix operations, even matrices containing mostly zeros are computationally costly to handle. The total number of arithmetic operations is therefore $(2(T-1)^2-1)(2S^3-S^2)$, increasing cubically with the number of states and quadratically with the number of re-encounter occasions. The computational cost becomes therefore substantial when the number of states and/or the number of re-encounter occasions is large. This motivates the development of a more efficient, optimized version of the multinomial model, which is what we do next.

2.2 Optimized multinomial implementation of the model

The optimization of the multistate mark-recapture model in the multinomial likelihood consists of different steps at different model stages.

<u>First step in optimization: reducing the dimensions of the m-array</u>

States represented in an m-array can be classified into four general categories:

1. **Unobservable states.** Individuals in these states can never be detected, and their observation probability is zero. Examples include “dead” in models without dead recoveries, “outside the study area” in models restricted to local re-encounters that

are used to estimate site fidelity, or “non-breeding” in models where only breeding individuals can be re-encountered (Schaub & Kéry, 2022).

2. **Transient observable states.** Individuals in these states can be detected but they deterministically transition to an unobservable state at the next capture occasion, from which they remain permanently unobservable thereafter. A typical example is “recently dead” in models that include dead recoveries: such dead individuals can be observed (i.e., “recovered”) once, during their occasion of death, but cannot be observed at any later occasions (Burnham, 1993).
3. **Age-structured observable states.** These states explicitly encode an age component (e.g. “alive, 1-year-old”) and individuals in them deterministically transition to the next age class (e.g., “alive, 2-year-old), or to an age-independent state at the following capture occasion. A particular case of age-structured observable states concerns states whose age class is juvenile (0-year-old). Such states are observable at release but structurally non-observable at subsequent occasions, because individuals necessarily age into the next class before the next encounter event (unless the juvenile age class spans more than one detection occasion, for example in models using monthly rather than annual occasions).
4. **Other observable states.** All remaining observable states fall into this category. Transitions from these states to other states are not deterministic.

The degree to which the m-array can be simplified will depend on the categories of the states in a model. First, as individuals are never released or re-encountered in (1) unobservable states, rows and columns corresponding to these states are filled with structural zeros.

Thus, both rows and columns corresponding to unobservable states carry no information about demographic parameters and can be removed (but note that this does not always imply that these states can be entirely ignored in the likelihood; see next section). Second, as individuals released in (2) transient observable states are never available for subsequent encounter, the corresponding rows are again filled with structural zeros across all encounter columns, except for the “never re-encountered” column. Rows corresponding to transient observable states therefore provide no information about demographic parameters and can be removed. Note that this is not the case for the columns corresponding to transient observable states, which must be retained, as individuals released in other states can be re-encountered in transient observable states; these columns thus provide information about demographic parameters. Furthermore, (3) age-structured observable states that differ only by age (e.g. “alive at 1 year old” vs. “alive at 2 years old”) can be simplified by exploiting the deterministic relationship between age at release and age at re-encounter. For each release cohort, the age of an individual at every subsequent re-encounter occasion is fully determined (e.g., an individual released at 1-year-old and re-encountered alive 2 years later must be a 3-year-old), such that age does not need to be treated as a separate observable state. Instead, age can be handled using an age matrix, as is generally done in state-space capture–recapture models (Schaub & Kéry, 2022). Columns corresponding to states that differ only by age therefore carry redundant information and can be combined by summation. Finally, columns corresponding to age-structured states whose age class is juvenile can be removed, since individuals cannot be re-encountered in a juvenile state

(except when the juvenile stage spans more than one detection occasion, in which case these columns should be retained).

We next illustrate with an example of a model with the following 4 states: 1) Alive as a juvenile, 2) Alive at one-year-old or older ("adult"), 3) Recently dead (dead between the last occasion and this occasion), and 4) Long dead (dead before the last occasion). This results in the m-array shown in Figure 2, that we need to simplify. In this example, we assume (as generally done) that dead recoveries occur only at the occasion of death. Consequently, the state "long dead" is unobservable. Therefore, we can delete the rows and columns that correspond to this state. Second, the state "recently dead" is a transient observable state, and thus we can delete the corresponding rows (but not columns). Third, the states "alive as juvenile" and "alive as adult" are observable and differ only by age. The corresponding columns can therefore be combined by merging them (alternatively, one can delete the columns corresponding to juveniles, which in our case is equivalent). Finally, we obtain the simplified m-array shown in Figure 2.

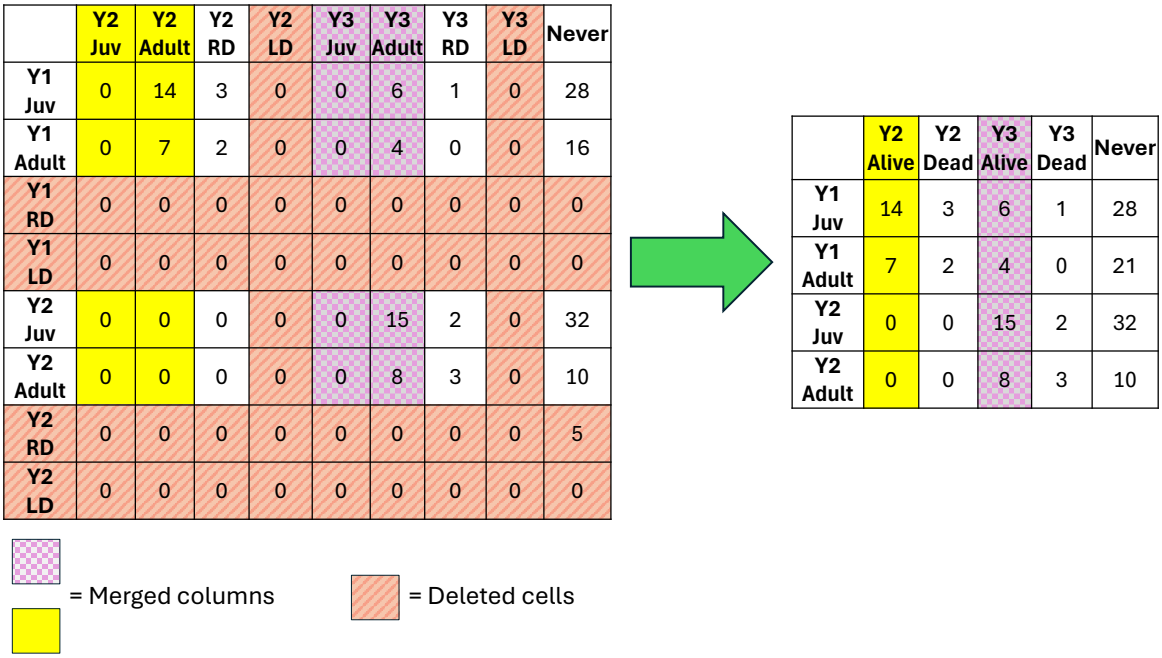

FIGURE 2. Example showing the dimension reduction of a simple m-array. Left is the m-array before, and right is the m-array after dimension reduction. Y1 refers to year 1, Y2 to year 2. In the conventional m-array, "Juv" denotes the "juvenile alive" state, "Adult" the "adult alive" state, "RD" the "recently dead" state, and "LD" the "long dead" state. In the optimized m-array, the two re-encounter states are "Alive" (irrespective of age) and "Dead", which corresponds to the former "recently dead" state. Column "Never" contains the numbers of individuals that were never re-encountered.

Let S' denote the number of states of release (i.e., the states whose rows have been retained in the simplified *m*-array) and S'' the number of observable and non-redundant states (i.e., the states whose columns were not removed and that remain distinct after merging). After

this dimensional reduction, the *m*-array contains $S'(T-1)$ rows and $S''(T-1)+1$ columns.

Second step in optimization: simplifying the vector of multinomial cell probabilities π

After reducing the dimensions of the m-array, the next step consists of computing the vector π', which contains the cell probabilities of the multinomial distributions in the optimized formulation and is therefore the equivalent of π in the conventional formulation. Here, the approach is different to the conventional formulation, which derives π through repeated multiplication of transition and observation matrices. This implicitly evaluates all state transitions, including the many whose probabilities are structurally zero. Because neither JAGS nor NIMBLE handles sparsity efficiently, performing these zero-per-zero multiplications is computationally costly. In the optimized approach, π' is calculated directly from the parameters, by considering all biologically feasible pathways (i.e., possible sequences of states across encounter occasions) between a release cohort *c'* and a re-encounter category *k'*. Second, unlike the conventional formulation, π' values are only computed for re-encounter occasions after each release cohort, eliminating additional unnecessary zero-calculations. All these modifications greatly reduce the number of arithmetic operations compared to the conventional approach. The full mathematical derivation of π' is provided in Appendix S2; the practical implementation is described stepwise in Tutorials S1 to S4, alongside annotated code and for both JAGS and NIMBLE.

Third step in the optimization: computation of the multinomial likelihood

Then, for each row $\boldsymbol{m}'_{c'}$ starting in column $D(c')$ of the simplified *m*-array, we can write:

$$\boldsymbol{m}'_{c',D(c'):K'} \sim Multinomial\ (R'_{c'}, \boldsymbol{\pi}'_{c'}), \qquad \text{(eqn. 2)}$$

where c' is the release cohort, $R'_{c'}$ is the total number of individuals released in cohort c' and $\boldsymbol{\pi}'_{c'} = (\pi'_{c'D,}, \pi'_{c',D+1}, \dots, \pi'_{c',K'}, \pi'_{c',\emptyset})$.

In contrast to the conventional multinomial formulation, we only compute and use columns from $D(c')$ onward (since earlier columns are impossible for cohort c'). This substantially reduces computation, especially for late release cohorts, which have few possible re-encounter occasions.

2.3 Evaluating model performance with data simulation

We simulated multistate CMR data under six scenarios differing in their underlying state structure. The first three scenarios represent classic multistate models where the states correspond to discrete sites. The last three scenarios represent the widely used live-resighting-dead-recovery models (Burnham, 1993) with an age structure, where states correspond to different age classes for live individuals, plus the typical recently dead and long-dead states (see below).

The three movement scenarios differed by the number of sites (2, 3, and 4 respectively), representing increasing levels of model complexity, and were designed to test how the optimization of the multinomial distribution performs as the complexity of the movement process (number of sites) increases. Individuals could move between sites and could also die, i.e., transition to the unobservable dead state, resulting in a total of 3, 4, and 5 states in the three scenarios, respectively. Each scenario included site-dependent survival and

resighting probabilities, as well as movement probabilities between sites. The parameter values used to generate the data are given in Table S1.

In the 3 age-structured scenarios, there were respectively 2 (juvenile, adult), 3 (juvenile, 1-year-old, adult), and 4 (juvenile, 1-year-old, 2-year-old, adult) age classes. Again, we increased complexity across scenarios to test how increasing age-structure complexity affects the efficiency of the optimized implementation of the multinomial model. In addition, every scenario included a "recently dead" state, when individuals could be recovered, and a "long dead" state, when they could no longer be recovered. The three scenarios thus comprised 4, 5, and 6 states, respectively. Each scenario included age-dependent survival and resighting probabilities, and a constant recovery probability across ages. The data-generating parameter values are provided in Table S2.

In all simulations, we modelled the survival and re-encounter of marked individuals over 15 years, with 80 individuals marked every year except for the last (1120 in total). For simplicity, no temporal variability was incorporated in any parameters of neither model. We performed 100 simulation replicates per scenario, yielding 100 independent sets of capture histories, which were then converted into m-arrays.

2.4 Models fitted to the simulated data

We fitted all simulated datasets described above with the conventional and the optimized approach to evaluate recovery of parameter values and their computational efficiency. In all models, we used vague priors – $\mathrm{Beta}(1,1)$ – for all parameters.

Multinomial movement models

We first implemented 3 conventional multinomial movement models that corresponded to the 3 movement scenarios. These models had respectively 3, 4, and 5 states, as the respective m-arrays. The implementation followed the general case shown in section 2.1.

We then optimized the implementation of these 3 multinomial movement models, starting by the m-array. Here, the m-array can only be simplified via the unobservable "dead" state. Rows and columns corresponding to this state can thus be removed from the m-array. There are no transient or age-structured states in movement models.

For defining the probability $\pi'_{c',k'}$ that an individual marked in cohort c' and release state $s' \in \{1, \dots, S'\}$, is first re-encountered in the time-state combination k', we needed to identify all possible pathways between the different observable states, each corresponding to an individual being alive in a given site and at a given occasion. In movement models, individuals may move to any site at any occasion, so all pathways involving live states are possible. However, pathways in which an individual transitions to the “dead” state and subsequently returns to an observable state are biologically impossible and were therefore removed from the computation of $\pi'_{c',k'}$. $\pi'_{c',\emptyset}$ and the multinomial likelihood were computed as shown in the general optimized case (section 2.2).

The complete workflow implemented in the code, from CMR data processing to fitting the optimized implementation of the model, is described in Tutorials S1 (NIMBLE) and S2 (JAGS).

Multinomial age-structured models

We implemented 3 conventional multinomial age-structured models that corresponded to the 3 age-structured scenarios. These models had respectively 4, 5, and 6 states. The implementation followed the general case (section 2.1).

In all these conventional models, the last state (long dead) is unobservable. As a first step of optimization, we removed rows and columns belonging to this state in the m-array. Furthermore, the state “recently dead” deterministically transitions to long dead and is therefore transient. Thus, we removed the rows of the m-array belonging to this fourth state. Finally, we had, depending on the age-structured models, 2 to 4 age-structured live states, which can be merged. Furthermore, the model includes a “juvenile alive” state in which individuals cannot be re-encountered; removing the corresponding columns from the m-array is in that case equivalent to merging them with the other age-specific states. Figure 2 (Section 2.2) provides a detailed example of how to handle the m-array of an age-dependent model with dead recoveries (with two age classes in that case).

The simplified m-arrays had all 28 fewer rows than the original (i.e., 28, 42 and 56 rows in total for respectively the models with 2, 3 and 4 age classes, compared with 56, 70 and 84 in the original arrays). Simplified m-arrays of all three models had only 29 columns (versus respectively 57, 71 and 85 for the original m-arrays). Therefore, increasing the number of age classes only enlarges the number of rows in the optimized m-array, whereas it enlarges both rows and columns in the conventional formulation.

In age-structured models, there is only one unique pathway for any given combination of release state s' and first re-encounter state r'. If an individual is re-encountered alive, it

must have survived from the occasion of release to the occasion of re-encounter and, if any intermediate occasions occurred between release and re-encounter, the individual must have remained unobserved on those occasions prior to being detected at re-encounter. Likewise, if an individual is re-encountered dead, this implies it has survived from release until the occasion preceding the re-encounter, was not detected on any intermediate occasions, died at the occasion of re-encounter, and was recovered dead at that occasion. Only these possible pathways were retained in the computation of $\pi'_{c\prime,k\prime}$ probabilities. In this computation, age dependence was handled by pre-computing an age matrix that gives the expected age class at each encounter occasion as a function of the release age class and time. $\pi'_{c\prime,\emptyset}$ and the multinomial likelihood were computed as shown in the general optimized case (section 2.2).

The complete workflow implemented in the code, from CMR data processing to fitting the optimized implementation of the model, is described in Tutorials S3 (NIMBLE) and S4 (JAGS).

2.5 Evaluation of the bias and computational efficiency

For each scenario, we fitted the two corresponding variants of the multinomial model (conventional and optimized) to each of the 100 simulated m-arrays, in both NIMBLE and JAGS (Plummer et al., 2003; de Valpine et al., 2017). On a MacBook Pro with an M4 Max processor, we ran 4 chains with 20,000 MCMC iterations each, with a burn-in of 1, and no thinning. For every run and primary (structural) parameter, we recorded the posterior mean and standard deviation, the potential scale reduction factor $\widehat{\mathrm{R}}$, and the effective sample size (ESS).

We calculated computational efficiency as the ESS of the least converged parameter (highest $\hat{R}$), divided by computation time (compilation plus MCMC running) in seconds, following prior work (e.g., Ponisio et al., 2020; Schaub & Badia-Boher, 2025).

In addition, to test for equivalence of the estimates of the conventional and optimized framework, we calculated four different metrics. First, we calculated Pearson correlation coefficients between posterior means of all parameters in every model. Second, we tested for any biases in the estimation of parameters by both model types against data-generating values. For every model m and primary parameter q, the mean relative bias (MRB) is

$$MRB_{m,j} = \frac{1}{H}\sum_{h=1}^{H}\frac{\hat{\theta}_{m,j,h} - \theta_{m,j}}{\theta_{m,j}} \quad \text{(eqn. 3)}$$

where H is the number of simulation replicates (here 100), $\hat{\theta}_{m,j,h}$ the estimated mean of the parameter j of model m for simulation replicate h, and $\theta_{m,j}$ is the true value of the parameter j in scenario m.

Finally, for each parameter, we computed coverage as the percentage of simulation replicates in which the true value fell within the 95% credible interval.

2.6 Case study

We used a large capture–recapture dataset of White Storks (*Ciconia ciconia*) from Germany to illustrate our optimized multinomial approach. Over 15 years (2008-2022), 12,552 individuals were marked and resighted in two regions (Bundesländer Saxony-Anhalt and Brandenburg). Given the scale, state-space capture-mark-recapture models become computationally too expensive, which motivates an analysis using an m-array.

To represent the life cycle of the white stork, we defined 13 states. They result from the combination of six age classes (juvenile, 1-year-old to 4-year-old, and ≥5-year-old) and the two regions treated as discrete sites, plus one state for dead individuals. The resulting m-array contained 182 rows and 183 columns.

Conventional implementation of the multinomial likelihood

We first fitted the conventional implementation of the multinomial likelihood. The 13 x 13 state-transition matrix (Figure S1) had the following parameters:

- Survival (ϕ) dependent on region, time, and three age classes: juvenile survival $\phi_{i,t,juv}$ for juveniles, immature survival $\phi_{i,t,imm}$ for 1 to 3-year-old individuals, and adult survival $\phi_{i,t,ad}$ for 4-year- and older individuals.
- Between-region movement (ψ) dependent on region (i.e., Saxony-Anhalt to Brandenburg was different from Brandenburg to Saxony-Anhalt) and age with 2 levels: juvenile movement $\psi_{i\rightarrow j,juv}$ for juveniles and adult movement $\psi_{i\rightarrow j,ad}$ for all other age classes.

These two parameters were multiplied in the state transition matrix. For example, the probability of transition from state "juvenile in Saxony-Anhalt" to state "1-year-old in Brandenburg" was $\phi_{SA,juv} \times \psi_{SA\rightarrow B,juv}$, where $\psi_{SA\rightarrow B,juv}$ is the juvenile movement from Saxony-Anhalt to Brandenburg.

We had a 13×13 observation matrix, the diagonal of which was populated with resighting probability (p) that was region-, time-, and age-dependent (the latter with five levels). Age-dependence was needed since white storks start breeding at 2 - 5 years old (Barbraud et al., 1999), and nonbreeders are less likely to be resighted.

We specified random time effects in survival and resighting probabilities, different and independent between regions. The multinomial cell probabilities (π) and the multinomial likelihood were then computed following the general rule for conventional multinomial frameworks.

Optimized implementation of the multinomial model

Rows and columns associated with the "long dead" state are removed from the m-array. There was no transient state, but all live states were age structured. Hence, columns associated with all states corresponding to the region "Saxony-Anhalt" were merged, and we did the same for those associated with all states corresponding to the region "Brandenburg", as the only difference between these region-specific states is the age. The optimized m-array had 168 rows and 29 columns.

The transitions between age classes are deterministic, while the transitions between sites are not. Therefore, possible pathways between two observable states include all pathways in which individuals may move between any sites, but only while progressing through age classes in the predefined order (juvenile to 1-year-old, 1-year-old to 2-year-old, etc.). Pathways in which an individual transitions to the "dead" state and subsequently returns to an observable state are impossible, as are pathways in which age classes do not progress in the predefined order. The cell probabilities $\pi'_{c',k'}$ were computed using only possible pathways. Then, the cell probabilities $\pi_{c',\emptyset}$ and the multinomial likelihood were then computed as in all optimized multinomial formulations.

For both the conventional and the optimized implementation, we ran (again on a MacBook Pro with an M4 Max processor) 4 chains with 20,000 iterations each, with a burn-in of 1 and no thinning. We ran the models in both NIMBLE and JAGS. We recorded the posterior mean and standard deviation of primary parameters, as well as the $\hat{R}$, and the ESS. We also calculated the computational efficiency of the conventional and optimized implementation as described previously.

3. Results

Coverage and bias across model types (age and movement), parameters, and fitting packages (JAGS and NIMBLE) were virtually identical between the conventional implementation of the multinomial model and the optimized version (Table S3, Table S4). Mean relative bias was generally negligible (<1%), and slightly larger in more parameterized models (e.g., 4-sites movement, 4-age-class), but remained always low (Table S3; max 10.17%). Coverage of 95% CRIs was near the nominal 95% level in all cases (Table S4).

Pearson correlation coefficients between posterior means were equal to 1 for all parameters (Fig. 3, Fig. S2-S5), strongly suggesting that our method indeed represents an alternative parameterization of the same model as with the traditional form of the m-array. Correlations between posterior standard deviations were also generally perfect (r = 1), with slightly lower values (r>0.8) for a small subset of parameters in the more complex age models (three and four age classes; Fig. S6-S9).

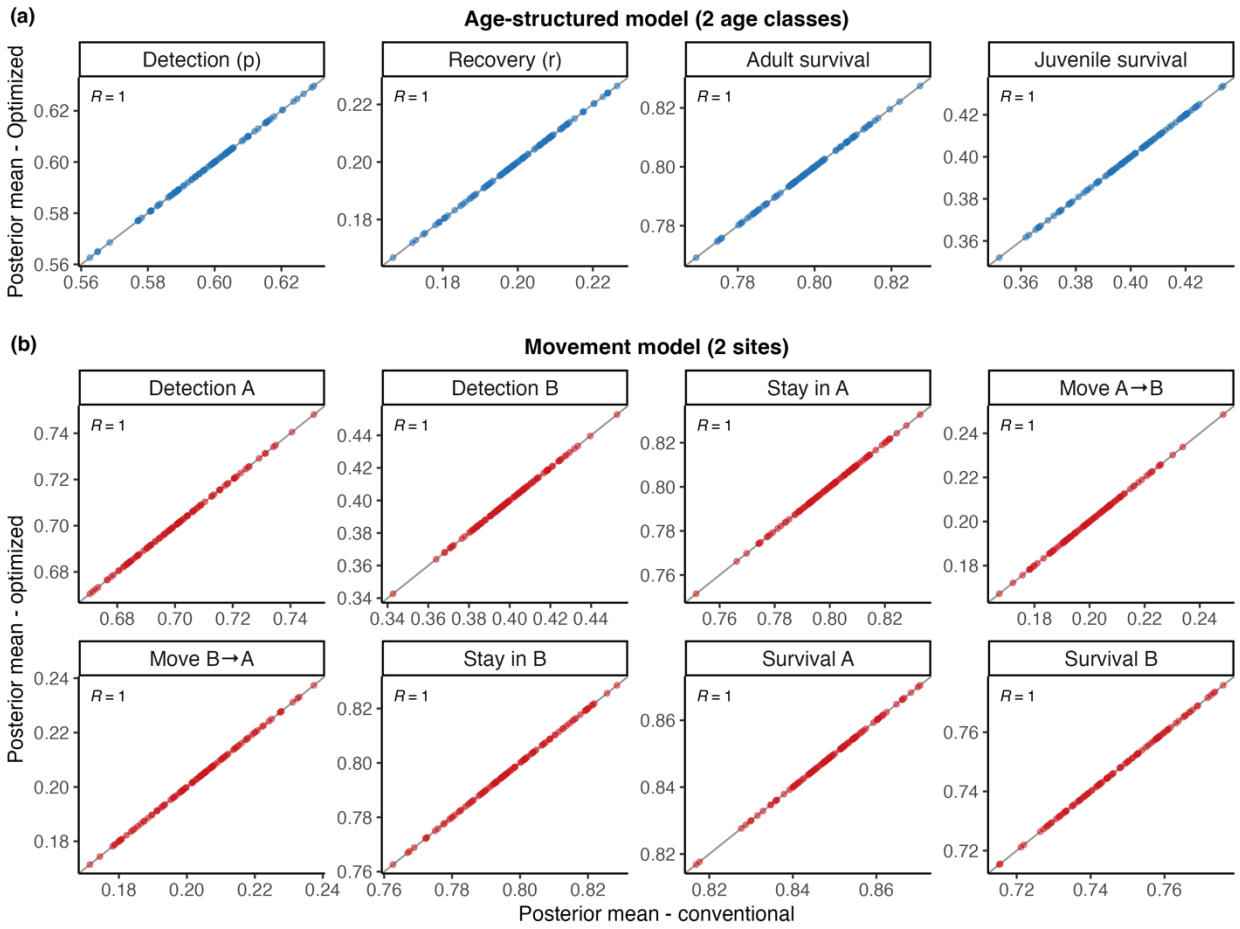

FIGURE 3. Correlations between posterior means of parameter estimates obtained from the conventional and optimized implementations in the 2-age-class and 2-site models in NIMBLE. Each point is defined by two corresponding posterior mean estimates from the same simulation replicate (n = 100 simulation replicates): the estimate obtained with the conventional implementation on the x-axis and the estimate obtained with the optimized implementation on the y-axis. The grey diagonal line represents the 1:1 line of perfect agreement. Pearson correlation coefficients (r) are shown in the upper left of each panel.

We report computational efficiencies as the ratio between the efficiency of the optimized divided by that of the conventional implementation (Table S5; Figure 4). In NIMBLE, in age-structured models, efficiency gains increased progressively with the number of age classes: models using the optimized implementation were 8.23 times more efficient (2.5^{th}-97.5^{th} percentiles: 7.98-8.55) for two age classes, 10.43 times (6.26-17.06) for three age classes, and 12.85 (7.96-19.49) for four age classes. In movement models, optimized models were 2.41 times more efficient (2.31-2.49) with two sites, 1.81 times faster (1.75-1.86) with three sites, and 1.84 times (1.77-1.90) with four sites.

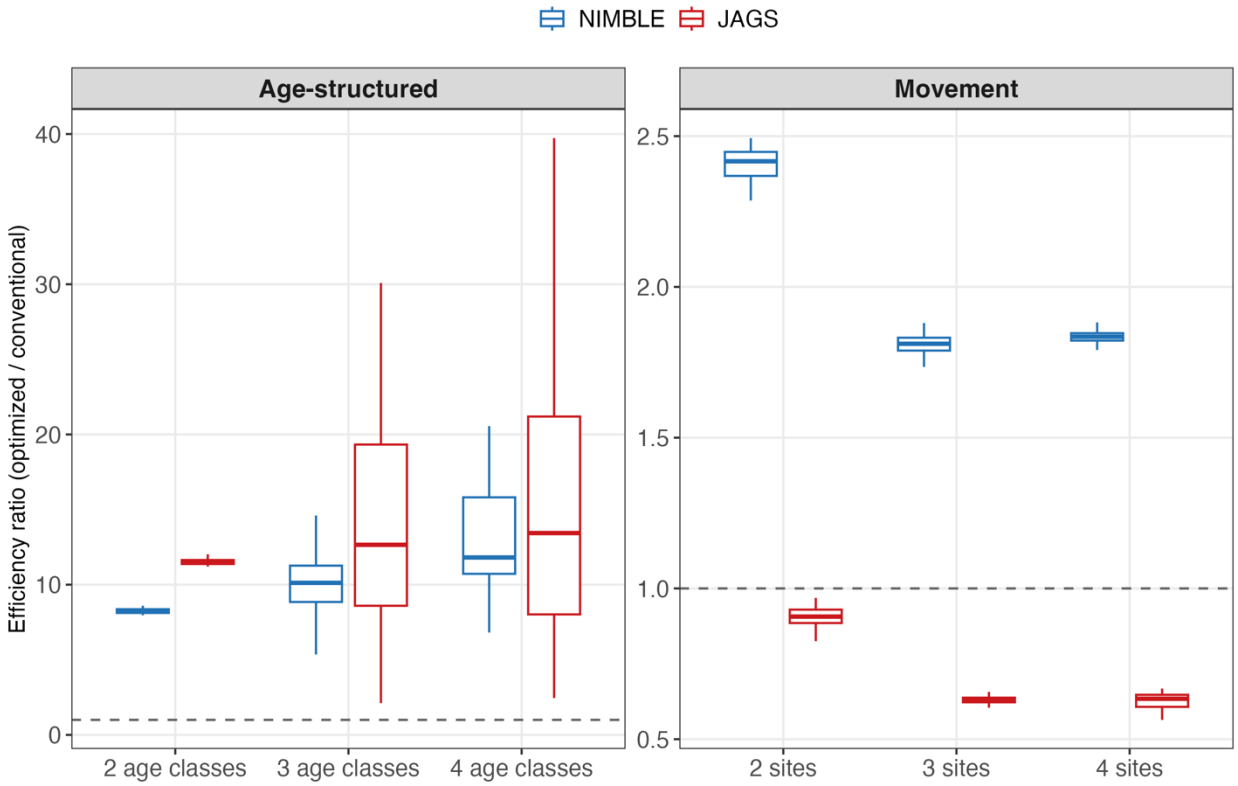

FIGURE 4. Computational efficiency gains of the optimized multinomial approach relative to the conventional multinomial by model type, structure, and fitting software. Each boxplot shows the distribution of efficiency ratios (optimized/conventional) across 100 simulation

replicates, where efficiency is defined as the effective sample size of the least-converged parameter divided by computation time (seconds).

Results in JAGS showed a similar pattern for age-structured models, but with larger computational gains that were again increasing with the number of age classes. Models using the optimized implementation were 11.47 times more efficient (10.62-12.27) for two age classes, 17.28 times (2.99-72.44) for three age classes, and 17.70 times (4.43-54.59) for four age classes. In contrast, models using the optimized approach in JAGS exhibited slightly lower computational efficiency than conventional formulations: two-site, three-site, and four-site models achieved relative efficiencies of 0.88 (0.70-0.95), 0.61 (0.47-0.65) and 0.62 (0.50-0.66) respectively.

Case study: German White Storks

Both the conventional and the optimized multinomial approach provided practically identical parameter estimates in both NIMBLE and JAGS (Table S6). Our results showed that apparent survival was slightly higher in Saxony-Anhalt than in Brandenburg, and that juvenile movement was more frequent from Saxony-Anhalt to Brandenburg than from Brandenburg to Saxony-Anhalt (Table S6).

In NIMBLE, the optimized implementation of the multinomial model ran 12.05 times faster than the conventional one. In terms of runtime only, the optimized approach took 1538 seconds, and the conventional approach 18545 seconds, to run 20,000 iterations. The

efficiency was also 11.27 times higher in the optimized multinomial approach (efficiency of 0.33) compared to the conventional one (efficiency of 0.03).

In JAGS, the optimized approach ran 5.65 times faster than the conventional one (1758 seconds compared to 9938 seconds for 20000 iterations), and was 5.06 times more efficient (efficiency of 0.29 vs 0.06).

4. Discussion

Our optimized implementation of the multinomial multistate CMR model consistently reproduced the estimates obtained with the conventional implementation while substantially improving computational efficiency, both in the simulations and in the case study and regardless of the engine (NIMBLE or JAGS). However, the magnitude of the efficiency gains depended strongly on model structure, with the largest improvements in models with age dependence, and less gains in models with only movement among sites.

The computational improvements of our optimized implementation arise from two complementary modifications to the conventional formulation. First, we reduce the dimensions of the m-array by removing the rows and columns that refer to structurally impossible transitions. In conventional implementations instead, the m-array is treated as a fixed object containing all possible combinations of release states, re-encounter states, and capture occasions, even if most transitions across them are impossible or deterministic (Schaub & Kéry, 2022). Second, we compute multinomial cell probabilities by calculating only the feasible pathways across states and over time. In the conventional formulation instead, calculations consider all state transitions at every time step, including many whose

probabilities are structurally zero. Because neither JAGS nor NIMBLE handles sparsity efficiently (i.e., arithmetic operations involving structural zeros are computationally costly), this results in substantial unnecessary computation. Crucially, this pathway-based calculation approach does not alter the multinomial likelihood itself. It is mathematically equivalent to the conventional formulation, as confirmed by the perfect correlations between posterior means across simulations and the case study. We note that pathway-based calculations that avoid structural zeros should in principle not be restricted to the multinomial likelihood, and could potentially be extended to other multistate CMR likelihood formulations, which may offer further computational gains worth exploring.

The extent to which our optimization approach is beneficial depends on the types of transitions in the model. In our study, age-structured models showed the largest improvements in computational efficiency, since age structures contain deterministic components in the state process, which can be removed. Because individuals age deterministically between capture occasions, their age at any future encounter is fully determined by their age at release and the time elapsed in-between. In conventional formulations, these age states appear separately in the m-array, and their transition probabilities – often zero – are still repeatedly evaluated through matrix multiplications. By collapsing such states and representing age classes through an additional matrix, the optimized implementation of the model reduces both the dimensionality of the m-array and the number of calculations required. Importantly, the gains tended to increase with model complexity. As the number of age classes increased, the number of rows, columns, and redundant calculations removed did so too. Such gains would most likely be greater in

models with more complex age structures (e.g., Véran & Lebreton, 2008; Lesley Szostek, 2014; Margalida et al., 2020).

In contrast, in movement models, transitions among sites are not deterministic, and hence, most state-time combinations remain possible. Therefore, only limited simplifications are possible. In our simulations, only the dead state could be removed, which resulted in relatively modest computational gains, and only in NIMBLE, not in JAGS. This difference between engines probably results from the different MCMC samplers that they use (Ponisio et al., 2020). Moreover, as model complexity increased (in terms of the number of sites), only the dead state could be removed, and hence the proportion of removable rows and columns in the m-array decreased, explaining why efficiency gains did not increase with model size in this case.

As stated above, Pearson correlations between posterior means of conventional and optimized formulations were always equal to 1, confirming that these two formulations are equivalent. Correlations between posterior standard deviations were also perfect, except for a few parameters in the three- and four-age-class age-structured models, where they remained high ($r > 0.8$). These small discrepancies reflect MCMC stochasticity rather than any structural difference between formulations. They likely result from our deliberate choice not to discard any iterations as burn-in, which, while appropriate for computing computational efficiency, leaves posterior variance estimates more sensitive to early sampling behaviour of the chains. Furthermore, they showed no systematic pattern of over- or underestimation, and in additional tests, we identified that correlations substantially improved with the number of MCMC iterations.

Our case study with German White storks comprised over 12,000 individuals, multiple age classes, movement among sites, dead-recoveries, and temporally-variable parameters with random effects. Despite the presence of site states, the age structure and dead-recovery process introduced deterministic relationships that allowed substantial simplification. As a result, the optimized implementation of the model was an order of magnitude more efficient than the conventional one in both NIMBLE and JAGS. These findings suggest that even when movement processes are included, large efficiency gains can still be achieved when models also contain deterministic state transitions. We anticipate that computational gains could be even greater in more complex models, where the number of states in conventional implementations may reach several dozens, and contain several m-array structures that can be simplified (e.g., Arévalo-Ayala et al., 2025, Muller et al., 2026).

The computational improvements obtained with the optimized approach translate into clear advantages for ecological analyses. Model development in Bayesian frameworks typically involves fitting multiple candidate models, assessing convergence, and comparing them (Hooten & Hobbs, 2015; ver Hoef & Boveng, 2015). Reducing runtime therefore greatly helps with the exploration of model space. In our case study, the conventional implementation of the model required in NIMBLE approximately 5 hours and 10 minutes to run 20,000 iterations, whereas only ca. 25 minutes were needed in the optimized version. Running ten candidate models under these conditions would require more than 50 hours with the conventional formulation but only about 4 hours and 15 minutes with the optimized one. Such gains in computational efficiency can also ease the fitting of more complex models that would otherwise be limited by prohibitive runtimes. This is particularly relevant for

integrated population models (IPMs) and other integrated models that include CMR data (Schaub & Kéry, 2022; Gregory et al., 2025), where the CMR component is frequently the computational bottleneck. In complex IPMs, achieving convergence can require days or even weeks of computation. By reducing the computational cost of the CMR likelihood, the optimized formulation can substantially improve the tractability of such models.

An inherent limitation of our approach is that the achievable computational gains depend strongly on the structure of the state process, as shown in our examples. Furthermore, model structures and state transitions are rarely identical across any model, and therefore, the specific m-array parts and model calculations that can be removed may strongly depend on the structure of the model. Hence, implementing the optimized formulation of the likelihood requires careful identification of states and transitions that can be removed, which may require additional conceptual and programming efforts compared to standard implementations. Developing automated tools to identify redundant elements of m-arrays and simplify likelihood calculations could help at expanding the method to less experienced users. A further limitation inherent to all multinomial CMR formulations is that summarizing data into m-arrays eliminates individual identity, preventing the inclusion of continuous individual covariates or individual random effects (Schaub & Badia-Boher, 2025). In situations where such terms are important, alternative formulations may be preferable (Yackulic et al., 2020, Schaub & Badia-Boher, 2025).

Overall, the optimized multinomial implementation of the likelihood provides a practical and accessible approach to speed up Bayesian multistate CMR models, while retaining the flexibility and multiple benefits of the hierarchical framework. By reducing computational

constraints, this approach enables the fitting of more complex models, the analysis of larger datasets, and more extensive exploration of ecological hypotheses. In an era of increasingly large ecological datasets, a growing popularity of Bayesian methods, and the need for testing more complex ecological theory, improving computational efficiency will be essential for fully exploiting the potential of modern demographic models.

References

Arévalo-Ayala, D. J., Real, J., Margalida, A., Badia-Boher, J. A., Mañosa, S., Durà, C., Aymerich, J., Jiménez, J., Martínez, J. M., & Hernández-Matías, A. (2025). Contrasting vital rate contributions across interconnected populations of a highly vagile avian scavenger: A multisite modelling approach. *Biological Conservation*, *311*, 111454.

Barbraud, C., Barbraud, J. C., & Barbraud, M. (1999). Population dynamics of the White Stork Ciconia ciconia in western France. *Ibis*, *141*(3), 469-479.

Besbeas, P., Freeman, S. N., Morgan, B. J. T., & Catchpole, E. A. (2002). Integrating Mark-Recapture-Recovery and Census Data to Estimate Animal Abundance and Demographic Parameters. *Biometrics*, *58*(3), 540–547.

Burnham, K.P. (1993). A theory for combined analysis of ring recovery and recapture data. In J.D. Lebreton & P.M. North (Eds.), *Marked individuals in the study of bird populations* (pp. 199-213). Birkhäuser Verlag.

Conn, P. B., & Cooch, E. G. (2009). Multistate capture–recapture analysis under imperfect state observation: An application to disease models. *Journal of Applied Ecology*, *46*(2), 486–492.

Cooch, E.G., & White, G.C. (2019). Program MARK – A Gentle Introduction, 19th Edn. Available online at: http://www.phidot.org/software/mark/docs/book/ (accessed March 27, 2026).

Gregory, K. A., Francesiaz, C., Jiguet, F., Crochet, P. A., Bocher, P., Düttmann, H., ... & Besnard, A. (2025). An integrative framework to combine migratory connectivity and demographic data. *Methods in Ecology and Evolution*, *16*(3), 596-610.

Hooten, M. B., & Hobbs, N. T. (2015). A guide to Bayesian model selection for ecologists. *Ecological Monographs*, *85*(1), 3–28.

Kéry, M., & Schaub, M. (2011). *Bayesian population analysis using WinBUGS: a hierarchical perspective*. Academic press.

Labuzzetta, C. J., Coulter, A. A., & Erickson, R. A. (2024). Comparing maximum likelihood and Bayesian methods for fitting hidden Markov models to multi-state capture-recapture data of invasive carp in the Illinois River. *Movement Ecology*, *12*(1), 2.

Lebreton, J.-D., Burnham, K. P., Clobert, J., & Anderson, D. R. (1992). Modeling Survival and Testing Biological Hypotheses Using Marked Animals: A Unified Approach with Case Studies. *Ecological Monographs*, *62*(1), 67–118.

Lebreton, J. D., Nichols, J. D., Barker, R. J., Pradel, R., & Spendelow, J. A. (2009). Modeling individual animal histories with multistate capture–recapture models. *Advances in ecological research*, *41*, 87-173.

Lesley Szostek, K., Schaub, M., & Becker, P.H. (2014). Immigrants are attracted by local pre-breeders and recruits in a seabird colony. *Journal of Animal Ecology*, 83, 1015-2024.

Margalida, A., Jiménez, J., Martínez, J. M., Sesé, J. A., García-Ferré, D., Llamas, A., Razin, M., Colomer, M., & Arroyo, B. (2020). An assessment of population size and demographic drivers of the Bearded Vulture using integrated population models. *Ecological Monographs*, *90*(3), e01414.

Muller, M. H., Ketwaroo, F. R., Fiedler, W., Geiter, O., Herrmann, C., & Schaub, M. (2026). Assessment of large-scale spatial variation in age-dependent survival and age at first breeding in a long-lived species. *Journal of Animal Ecology*. Advance online publication. https://doi.org/10.1111/1365-2656.70291.

Nichols, J. D., Sauer, J. R., Pollock, K. H., & Hestbeck, J. B. (1992). Estimating Transition Probabilities for Stage-Based Population Projection Matrices Using Capture-Recapture Data. *Ecology*, *73*(1), 306–312.

Plummer, M. (2003). JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling. In *Proceedings of the 3rd international workshop on distributed statistical computing*. Technische Universität Wien.

Ponisio, L. C., De Valpine, P., Michaud, N., & Turek, D. (2020). One size does not fit all: Customizing MCMC methods for hierarchical models using NIMBLE. *Ecology and Evolution*, *10*(5), 2385–2416.

Schaub, M., & Abadi, F. (2011). Integrated population models: A novel analysis framework for deeper insights into population dynamics. *Journal of Ornithology*, *152*(Suppl 1), 227–237.

Schaub, M., & Badia-Boher, J. A. (2025). Comparison of Bayesian Models to Estimate Survival From Dead-Recovery Alone and Together With Live-Encounter Data: Challenges and Opportunities. *Ecology and Evolution*, *15*(6), e71517.

Schaub, M., & Kéry, M. (2022). *Integrated population models: Theory and ecological applications with R and JAGS*. Academic Press.

de Valpine, P., Turek, D., Paciorek, C.J., Anderson-Bergman, C., Temple Lang, D., & Bodik, R. (2017). Programming with models: writing statistical algorithms for general model structures with NIMBLE. *Journal of Computational and Graphical Statistics*, 26, 403-413.

Ver Hoef, J. M., & Boveng, P. L. (2015). Iterating on a single model is a viable alternative to multimodel inference. *The Journal of Wildlife Management*, *79*(5), 719–729.

Véran, S., & Lebreton, J. (2008). The potential of integrated modelling in conservation biology: A case study of the black-footed albatross (*Phoebastria nigripes*). *Canadian Journal of Statistics*, *36*(1), 85–98.

Yackulic, C. B., Dodrill, M., Dzul, M., Sanderlin, J. S., & Reid, J. A. (2020). A need for speed in Bayesian population models: A practical guide to marginalizing and recovering discrete latent states. *Ecological Applications*, *30*(5), e02112.

Optimizing the Multinomial Likelihood - Movement Models - NIMBLE

Anonymized

Load packages and simulate the data

```
require (IPMbook)
require (nimble)
```

Let's simulate a dummy mark-recapture dataset to be used as an example, called y. Fix a seed first.

```
set.seed(2026)
```

Now simulate the data. Recall: movement model, two sites.

```
# Define data-generating values.

phiA <- 0.85 # Survival at site A
phiB <- 0.75 # Survival at site B
psiAB <- 0.2 # Movement from site A to site B
psiBA <- 0.2 # Movement from site B to site A
pA <- 0.7 # Detection probability at site A
pB <- 0.4 # Detection probability at site B

nyears <- 15 # Number of years
n.states <- 3 # Number of states (1: Alive at site A; 2: Alive at site B; 3:
Dead)
n.obs <- 3 # Number of observations (1: Seen Alive at A; 2: Seen Alive at B;
3: Not Seen)
nmarked <- 80 # Number of marked individuals per occasion/year

# Define vector of first encounters
f <- rep(1:(nyears-1), each=nmarked)
nind <- length(f) # Total number of individuals

# Define transition matrices.
# These are 4-dimensional matrices, with
# Dimension 1: state of departure
# Dimension 2: state of arrival
# Dimension 3: individual
# Dimension 4: time


# 1. State process matrix
```

```
PSI.STATE <- array(NA, dim = c(n.states, n.states, nind, nyears-1))

for(i in 1:nind){
  for(t in 1:(nyears-1)){

    PSI.STATE[,,i,t] <- matrix(c(

      phiA*(1-psiAB), phiA*psiAB, 1-phiA,
      phiB*psiBA, phiB*(1-psiBA), 1-phiB,
      0,0,1), nrow = n.states, byrow = TRUE)
  }
}

#2. Observation process matrix

PSI.OBS <- array(NA, dim = c(n.states, n.obs, nind, nyears-1))
for(i in 1:nind){
  for(t in 1:(nyears-1)){

    PSI.OBS[,,i,t] <- matrix(c(

      pA, 0, 1-pA,
      0, pB, 1-pB,
      0, 0, 1), nrow = n.states, byrow = TRUE

    )

  }
}

# State or ecological process
# Simulate true system state
z <- array(NA, dim=c(nind, nyears)) # Create empty true state matrix.
# Initial conditions: all individuals alive at f(i), but start either at site
 A or at B.
initial.state <- c(rep(1, round(nind/2)), rep(2, round(nind/2)))
for (i in 1:nind){
  z[i,f[i]] <- initial.state[i]
}

# Propagate true state process forwards via transition rule
for (i in 1:nind){
  for (t in (f[i]+1):nyears){
    departure.state <- z[i,t-1]
    arrival.state <- which(rmultinom(1,1, PSI.STATE[departure.state,,i,t-1])=
=1)
    z[i,t] <- arrival.state
  } #t
```

```
} #i

# Observation process: simulate observations using observation matrix
y <- array(3, dim=c(nind, nyears))
for (i in 1:nind){
  y[i,f[i]] <- z[i,f[i]]
  for (t in (f[i]+1):nyears){
    true.state <- z[i,t]
    observed.state <- which(rmultinom(1,1, PSI.OBS[true.state,,i,t-1])==1)
    y[i,t] <- observed.state
  } #t
} #i
```

Have a look at the first 6 rows of the mark-recapture dataset:

```
head (y)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,1
4]
[1,]    1    3    3    3    3    3    3    3    3     3     3     3     3
 3
[2,]    1    1    1    1    1    1    3    2    3     3     2     3     3
 3
[3,]    1    3    1    1    1    1    2    2    3     2     2     3     3
 3
[4,]    1    1    1    1    1    2    2    1    1     3     3     3     3
 3
[5,]    1    1    3    3    3    3    3    3    3     3     3     3     3
 3
[6,]    1    1    3    3    3    3    3    3    3     3     3     3     3
 3
     [,15]
[1,]     3
[2,]     2
[3,]     3
[4,]     3
[5,]     3
[6,]     3
```

The observation codes have the following meanings:

- 1 = detected alive at site A
- 2 = detected alive at site B
- 3 = not observed (this state is not observable)

Generate the conventional m-array

Here we transform the encounter histories into a conventional m-array. We need to define the 3 states of the model first:

- 1 = alive in site A
- 2 = alive in site B
- 3 = dead

Replace observation code “not observed” (3) with a 0. We do this because the function “marray” from package IPMbook needs the “non-observed” event to be a 0.

```
y[y == 3] <- 0
```

Set how many states are unobservable, and create the m-array using function “marray” from package IPMbook. Recall that there is one unobservable state: state 3 (dead), since in this model there are no dead recoveries.

```
unobs <- 1
marr <- marray(y, unobs = unobs)
```

Have a look at the m-array:

```
head (marr)
```

```
        recaptured
released Y2.S1 Y2.S2 Y2.S3 Y3.S1 Y3.S2 Y3.S3 Y4.S1 Y4.S2 Y4.S3 Y5.S1 Y5.S2
   Y1.S1    38     2     0    12     3     0     3     0     0     0     2
   Y1.S2     0     0     0     0     0     0     0     0     0     0     0
   Y1.S3     0     0     0     0     0     0     0     0     0     0     0
   Y2.S1     0     0     0    47     8     0    16     4     0     5     4
   Y2.S2     0     0     0     0     0     0     2     0     0     0     0
   Y2.S3     0     0     0     0     0     0     0     0     0     0     0
        recaptured
released Y5.S3 Y6.S1 Y6.S2 Y6.S3 Y7.S1 Y7.S2 Y7.S3 Y8.S1 Y8.S2 Y8.S3 Y9.S1
   Y1.S1     0     0     0     0     0     0     0     0     0     0     0
   Y1.S2     0     0     0     0     0     0     0     0     0     0     0
   Y1.S3     0     0     0     0     0     0     0     0     0     0     0
   Y2.S1     0     0     0     0     0     0     0     0     0     0     0
   Y2.S2     0     0     0     0     0     0     0     0     0     0     0
   Y2.S3     0     0     0     0     0     0     0     0     0     0     0
        recaptured
released Y9.S2 Y9.S3 Y10.S1 Y10.S2 Y10.S3 Y11.S1 Y11.S2 Y11.S3 Y12.S1 Y12.S2
   Y1.S1     0     0      0      0      0      0      0      0      0      0
   Y1.S2     0     0      0      0      0      0      0      0      0      0
   Y1.S3     0     0      0      0      0      0      0      0      0      0
   Y2.S1     0     0      0      0      0      0      0      0      0      0
   Y2.S2     0     0      0      0      0      0      0      0      0      0
   Y2.S3     0     0      0      0      0      0      0      0      0      0
        recaptured
```

```
released Y12.S3 Y13.S1 Y13.S2 Y13.S3 Y14.S1 Y14.S2 Y14.S3 Y15.S1 Y15.S2 Y15.S
3
   Y1.S1      0      0      0      0      0      0      0      0      0
0
   Y1.S2      0      0      0      0      0      0      0      0      0
0
   Y1.S3      0      0      0      0      0      0      0      0      0
0
   Y2.S1      0      0      0      0      0      0      0      0      0
0
   Y2.S2      0      0      0      0      0      0      0      0      0
0
   Y2.S3      0      0      0      0      0      0      0      0      0
0
        recaptured
released never
   Y1.S1    20
   Y1.S2     0
   Y1.S3     0
   Y2.S1    34
   Y2.S2     0
   Y2.S3     0
```

1. First part in the optimization process: simplify the m-array

Step 1A: Unobservable states

Here, we have the state 3 (dead) which is unobservable. We can therefore remove rows and columns belonging to state 3 as they do not provide information for inference.

```
marr.opti <- marr[!grepl("S3", rownames(marr)), !grepl("S3", colnames(marr))]
```

As opposite to age models, further simplification of the m-array is not possible.

Let's have a look at the resulting simplified m-array:

```
head (marr.opti)

        recaptured
released Y2.S1 Y2.S2 Y3.S1 Y3.S2 Y4.S1 Y4.S2 Y5.S1 Y5.S2 Y6.S1 Y6.S2 Y7.S1
   Y1.S1    38     2    12     3     3     0     0     2     0     0     0
   Y1.S2     0     0     0     0     0     0     0     0     0     0     0
   Y2.S1     0     0    47     8    16     4     5     4     0     0     0
   Y2.S2     0     0     0     0     2     0     0     0     0     0     0
   Y3.S1     0     0     0     0    64    12    15     8     4     3     3
   Y3.S2     0     0     0     0     0     2     0     1     0     1     0
        recaptured
released Y7.S2 Y8.S1 Y8.S2 Y9.S1 Y9.S2 Y10.S1 Y10.S2 Y11.S1 Y11.S2 Y12.S1
   Y1.S1     0     0     0     0     0      0      0      0      0      0
   Y1.S2     0     0     0     0     0      0      0      0      0      0
   Y2.S1     0     0     0     0     0      0      0      0      0      0
```

```
   Y2.S2     0     0     0     0     0     0     0     0     0     0
   Y3.S1     1     2     0     0     0     0     0     0     0     0
   Y3.S2     0     0     0     0     0     0     0     0     0     0
         recaptured
released Y12.S2 Y13.S1 Y13.S2 Y14.S1 Y14.S2 Y15.S1 Y15.S2 never
   Y1.S1      0      0      0      0      0      0      0    20
   Y1.S2      0      0      0      0      0      0      0     0
   Y2.S1      0      0      0      0      0      0      0    34
   Y2.S2      0      0      0      0      0      0      0     0
   Y3.S1      0      0      0      0      0      0      0    27
   Y3.S2      0      0      0      0      0      0      0     7
```

It is still smaller than the non-simplified m-array

```
compar_dim <- data.frame(row_number = c(nrow(marr), nrow(marr.opti)), col_num
ber = c(ncol(marr), ncol(marr.opti)))
rownames(compar_dim) = c("Original m-array", "Simplified m-array")

compar_dim
```

```
                  row_number col_number
Original m-array          42         43
Simplified m-array        28         29
```

Step 1B: Create vector of first encounters "year_first"

This is a vector giving the year of release for each row in the m-array. Formulating it prevents computation of state-year combinations before the state-year of release of the individual. It will be used in the calculation of π and in the multinomial likelihood.

```
row_names <- rownames(marr.opti)
year_first <- as.numeric(sub("Y([0-9]+)\\.S[0-9]+", "\\1", row_names))
```

It looks like this:

```
year_first
```

```
 [1]  1  1  2  2  3  3  4  4  5  5  6  6  7  7  8  8  9  9 10 10 11 11 12 12
13
[26] 13 14 14
```

Create a vector storing the site of first capture for each row of the m-array. This allows calculating the pathway of each cohort starting from their site of release.

```
site_of_capture <- as.numeric(sub("Y[0-9]+\\.S([0-9]+)", "\\1", row_names))
```

Which looks like:

```
site_of_capture
```

```
 [1] 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
```

Define the number of years:

```
nyears <- 15
```

Define the number of states in which individuals are released and re-encountered. Here it is 2 states in the two:

- Alive at site A
- Alive at site B

```
# Number of states of release
ns.rel <- 2

# Number of states of re-encounter
ns.obs <- 2
```

2. Second part in the optimization process: write the model

We will first show how to construct the model section by section, and next, we will provide the whole model.

Step 2A) Assemble the data

Pass lists for the data and the constants.

```
my.constants <- list(nyears = nyears,
                     ns.rel = ns.rel,
                     ns.obs = ns.obs,
                     year_first = year_first,
                     site_of_capture = site_of_capture)

my.data <- list(marr = marr.opti,
                rel = rowSums(marr.opti))
```

Note: rel is the number of individuals released in each row of the m-array, i.e., the sum of the rows of the m-array. It is also used in conventional multinomial models.

Step 2B) Create the model and define the priors

The first part of the model does not change: that where we define the priors of our parameters.

```
msopti <- nimbleCode({

    model{

  # Define the priors

  # Survival
```

```
   sA ~ dbeta(1,1) # Vague prior for survival at site A
   sB ~ dbeta(1,1) # Vague prior for survival at site B

  # Movement: only between two sites

  psiA[1] ~ dbeta(1,1) # Vague prior for prob. of staying in site A
  psiA[2] <- 1-psiA[1] # Prob. of moving from site A to B is 1 - prob. of sta
ying in site A

  psiB[1] ~ dbeta(1,1) # Vague prior for prob. of moving from site B to A
  psiB[2] <- 1-psiB[1] # Prob. of staying in site B is 1 - prob. of moving fr
om site B to A

  # Detection probabilities

  pA ~ dbeta(1,1) # Vague prior for the detection probability in site A
  pB ~ dbeta(1,1) # Vague prior for the detection probability in site B
```

We additionally set all parameters in convenient structures for likelihood calculations.

First, we set all movement parameters into a matrix.

Second, we set all survival and detection parameters into a matrix with the number of columns equal to the number of years (years) minus one. This second step does not need to be done, as our parameters are time-constant, but we do it anyways so that it is easier to switch to a model with time-dependent parameters without needing to modify the likelihood.

```
# Define the movement matrix. Can be done with a loop if more sites

  mov [1,1] <- psiA[1] # probability of staying in A
  mov [1,2] <- psiA[2] # probability of moving from A to B
  mov [2,1] <- psiB[1] # probability of moving from B to A
  mov [2,2] <- psiB[2] # probability of staying in B

# Define survival as matrix (dependent on site and time, in case we want time
-dependence)

  for (t in 1:(nyears-1)) {

    s [1,t] <- sA # survival at A
    s [2,t] <- sB # survival at B

  }

 # Define detection probabilities as a matrix (dependent on site and time, in
 case we want time-dependence)
```

```
for (t in 1:(nyears-1)) {

  p [1,t] <- pA # detection at A
  p [2,t] <- pB # detection at B

}
```

Step 2C) Define π' and the optimized multinomial likelihood

Here we define the vector π', the equivalent to the vector π ("pi") in conventional multinomial models. Vector π' stores the probability for each cohort to be alive and detected in each site. The probabilities stored in π' are the cell probabilities used in the multinomial likelihood. Importantly, as opposite to the conventional implementation, in the optimized approach we calculate π' only from the possible pathways across state-year combinations of each cohort. This approach prevents calculation of structurally impossible transitions and results in an increase in computational efficiency.

As an auxiliary object to π' for pathway calculation purposes, we also define U. This object stores the probability of having survived and not having been observed before a subsequent observation.

2C1) Calculations for first year after release

Open the loop: "For each line in the new m-array, except the lines that indicate releases in the penultimate year":

```
for (t in 1:((nyears-2)*ns.rel)) {
```

First we perform all the calculations for the first year after release in the m-array.

The first year after release, the probability **U** to be alive and not detected in all sites is the product of the probabilities of:

1. Having survived in the site of release on the year of release.
2. Either have moved to another site or remain in the site of release
3. Not being detected the first year after release.

```
 U [t,(year_first[t]*ns.obs-ns.obs+1):(year_first[t]*ns.obs)] <- s[site_of_ca
pture [t], year_first[t]] * mov[site_of_capture[t],1:ns.obs] * (1-p [1:ns.obs,
 year_first[t]])
```

The first year after release, the probability **π'** to be alive and detected in each site is the product of the probabilities of:

1. Having survived in the site of release on the year of release.

2. Either have moved to another site or remain in the site of release

3. Being detected the first year after release.

```
 pi [t,(year_first[t]*ns.obs-ns.obs+1):(year_first[t]*ns.obs)] <-  s[site_of_
capture[t], year_first[t]] *  mov[site_of_capture[t],1:ns.obs] * p [1:ns.obs,
 year_first[t]]
```

2C2) Calculations for the following years except for releases in the penultimate year

Now we do the calculations for the following years after release in the m-array (up to the penultimate year). For this we open a nested loop.

```
for (n in (year_first[t]+1):(nyears-1)) {
```

We first calculate U for any given year t after the first year after release, up to the last year. Recall that here, for any given time after the first year after release, U is the product of the probabilities of:

1. Having survived between the year of release and t-1.

2. Being in each site at t without having been detected at any year before t.

3. Surviving from t-1 to t in each site.

4. Moving from a site to another, or remaining at the original site, between t-1 and t.

5. Not being detected in each site at t

Note that here the brackets are extremely important. You need to multiply U by survival before you multiply by the movement matrix. Otherwise, it will be considered that some individuals can move, then die.

```
  U [t,(n*ns.obs-ns.obs+1):(n*ns.obs)] <- (U [t,((n-1)*ns.obs-ns.obs+1):((n-1)
*ns.obs)] * s [1:ns.obs, n]) %*%      mov[1:ns.obs,1:ns.obs] * (1-p [1:ns.obs,
 n])
```

We then calculate π', which for any given time t after the first year after release, is the product of the probabilities of:

1. Having survived between the year of release and t-1.

2. Being in each site at t without having been detected in any previous year.

3. Surviving from t-1 to t in each site.

4. Moving from a site to another, or remaining at the original site, between t-1 and t.

5. Being detected in each site at t.

Again pay attention to the brackets here for the same reason as stated before.

```
pi [t,(n*ns.obs-ns.obs+1):(n*ns.obs)] <- (U [t,((n-1)*ns.obs-ns.obs+1):((n-1)
*ns.obs)] * s [1:ns.obs, n]) %*%   mov[1:ns.obs,1:ns.obs] * p [1:ns.obs, n]

    }
  }
```

Note we just closed the brackets of the two loops.

2C3) Calculations for releases in the penultimate year

We now calculate π' for those lines that correspond to releases in the penultimate year in the m-array.

For these lines, we only need to calculate probabilities for the first year after release, which is the last year of the m-array.

Note we do not need to compute U for these lines, because we only need π' for the multinomial likelihood and U is only used to compute π' for the years after the first year after release.

Start by defining a loop for each line in the new m-array that corresponds to releases in the penultimate year. Follow the same logic as explained above for the first year after release.

```
for (t in ((nyears-2)*ns.rel+1):((nyears-1)*ns.rel)) {

pi [t,(year_first[t]*ns.obs-ns.obs+1):(year_first[t]*ns.obs)] <- s[site_of_ca
pture[t], year_first[t]] * mov[site_of_capture[t],1:ns.obs] * p [1:ns.obs, ye
ar_first[t]]

}
```

2C4) Calculate π' for never being re-encountered after release

The probability of being never re-encountered is 1 - the probability of being re-encountered in any of the sites and any of the years after release.

```
  for (t in 1:((nyears-1)*ns.rel)) {

  pi [t,1+(ns.obs*(nyears-1))] <- 1-sum (pi [t,(year_first[t]*ns.obs-(ns.obs-
1)):(ns.obs*(nyears-1))])
```

2C5) Define the multinomial likelihood

We can do so within the same loop.

```
# Define multinomial likelihood

    marr [t,(year_first[t]*ns.obs-(ns.obs-1)):(nyears*ns.obs-(ns.obs-1))] ~ d
multi (pi [t,(year_first[t]*ns.obs-(ns.obs-1)):(nyears*ns.obs-(ns.obs-1))], r
```

```
el [t])

  }
```

And end the model.

```
})
```

3. Third part in the optimization process: run the model

Here's the whole model.

```
msopti <- nimbleCode({

  #Survival

  sA ~ dbeta(1,1)
  sB ~ dbeta(1,1)

  #Movement: only between two sites

  psiA[1] ~ dbeta(1,1)
  psiA[2] <- 1-psiA[1]

  psiB[1] ~ dbeta(1,1)
  psiB[2] <- 1-psiB[1]

  #Detection probabilities

  pA ~ dbeta(1,1)
  pB ~ dbeta(1,1)

  # Define the movement matrix

  mov [1,1] <- psiA[1]
  mov [1,2] <- psiA[2]
  mov [2,1] <- psiB[1]
  mov [2,2] <- psiB[2]

  # Set survival in a time- and site-dependent matrix

  for (t in 1:(nyears-1)) {

    s [1,t] <- sA
    s [2,t] <- sB

  }

  # Define detection probabilities in a time- and site-dependent matrix
```

```
  for (t in 1:(nyears-1)) {

    p [1,t] <- pA
    p [2,t] <- pB

  }

  # Define π' (named 'pi' in the code)
  # Calculate π' for first year after release

  for (t in 1:((nyears-2)*ns.rel)) {
    U [t,(year_first[t]*ns.obs-ns.obs+1):(year_first[t]*ns.obs)] <- s[site_of
_capture [t], year_first[t]] * mov[site_of_capture[t],1:ns.obs] * (1-p [1:ns.
obs, year_first[t]])

    pi [t,(year_first[t]*ns.obs-ns.obs+1):(year_first[t]*ns.obs)] <- s[site_o
f_capture[t], year_first[t]] * mov[site_of_capture[t],1:ns.obs] * p [1:ns.obs,
 year_first[t]]

        # Calculate π' for following years until penultimate year

    for (n in (year_first[t]+1):(nyears-1)) {

      U [t,(n*ns.obs-ns.obs+1):(n*ns.obs)] <- (U [t,((n-1)*ns.obs-ns.obs+1):
((n-1)*ns.obs)] * s [1:ns.obs, n]) %*%        mov[1:ns.obs,1:ns.obs] * (1-p
[1:ns.obs, n])

      pi [t,(n*ns.obs-ns.obs+1):(n*ns.obs)] <- (U [t,((n-1)*ns.obs-ns.obs+1):
((n-1)*ns.obs)] * s [1:ns.obs, n]) %*%     mov[1:ns.obs,1:ns.obs] * p [1:ns.o
bs, n]

    }
  }

      # Calculate π' for the penultimate year

  for (t in ((nyears-2)*ns.rel+1):((nyears-1)*ns.rel)) {

    pi [t,(year_first[t]*ns.obs-ns.obs+1):(year_first[t]*ns.obs)] <- s[site_o
f_capture[t], year_first[t]] * mov[site_of_capture[t],1:ns.obs] * p [1:ns.obs,
 year_first[t]]

  }

  # Calculate π' for the penultimate year for individuals that were never re-
encountered
```

```
  for (t in 1:((nyears-1)*ns.rel)) {

    pi [t,1+(ns.obs*(nyears-1))] <- 1-sum (pi [t,(year_first[t]*ns.obs-(ns.ob
s-1)):(ns.obs*(nyears-1))])

    # Define multinomial likelihood

    marr [t,(year_first[t]*ns.obs-(ns.obs-1)):(nyears*ns.obs-(ns.obs-1))] ~ d
multi (pi [t,(year_first[t]*ns.obs-(ns.obs-1)):(nyears*ns.obs-(ns.obs-1))], r
el [t])

  }
}
)
```

Define initial values.

```
inits <- function(){list(
  sA = rbeta(1,1,1),
  sB = rbeta(1,1,1),
  pA = rbeta(1,1,1),
  pB = rbeta(1,1,1),
  psiA = rbeta(2,1,1),
  psiB = rbeta(2,1,1)
)}
```

Set parameters to save and MCMC settings

```
parameters = c("sA", "sB",  "pA", "pB", "psiA", "psiB")

ni <- 20000 # number of iterations
nb <- 1 # number of burn-in iterations
nc <- 4 # number of chains
nt <- 1 # thinning rate
```

Build, compile, and run the model

We could also do it with the single-line call of Nimble, but we will do it using the line-by-line call, which is equivalent.

```
# Build and compile the model
mod <- nimbleModel(code = msopti, constants = my.constants, data = my.data, i
nits = inits())

Defining model

Building model

Setting data and initial values
```

```
Running calculate on model
  [Note] Any error reports that follow may simply reflect missing values in m
odel variables.

Checking model sizes and dimensions

  [Note] This model is not fully initialized. This is not an error.
         To see which variables are not initialized, use model$initializeInfo
().
         For more information on model initialization, see help(modelInitiali
zation).

Cmod <- compileNimble(mod)

Compiling
  [Note] This may take a minute.
  [Note] Use 'showCompilerOutput = TRUE' to see C++ compilation details.

# Build and compile the MCMC
modelConf <- configureMCMC(mod, monitors = parameters)

===== Monitors =====
thin = 1: pA, pB, psiA, psiB, sA, sB
===== Samplers =====
RW sampler (6)
  - sA
  - sB
  - psiA[]  (1 element)
  - psiB[]  (1 element)
  - pA
  - pB

modelMCMC <- buildMCMC(modelConf)
CmodelMCMC <- compileNimble(modelMCMC, project = mod)

Compiling
  [Note] This may take a minute.
  [Note] Use 'showCompilerOutput = TRUE' to see C++ compilation details.

# Run the MCMC
mcmc.out <- runMCMC(CmodelMCMC, niter = ni, nburnin = nb, thin = nt, nchains
= nc, progressBar = TRUE, samples = TRUE, summary = TRUE)

running chain 1...

|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|

running chain 2...

|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
running chain 3...

|-------------|-------------|-------------|-------------|
|--------------------------------------------------------|

running chain 4...

|-------------|-------------|-------------|-------------|
|--------------------------------------------------------|
```

And finally, show the summary for all chains.

```
print (mcmc.out$summary$all.chains)

             Mean    Median     St.Dev. 95%CI_low 95%CI_upp
pA      0.6782068 0.6784170 0.016872349 0.6457028 0.7101494
pB      0.4457107 0.4452211 0.021356548 0.4058069 0.4886930
psiA[1] 0.8093580 0.8096740 0.013022600 0.7828006 0.8339381
psiA[2] 0.1906420 0.1903260 0.013022600 0.1660619 0.2171994
psiB[1] 0.2151574 0.2146642 0.014561269 0.1880767 0.2451955
psiB[2] 0.7848426 0.7853358 0.014561269 0.7548045 0.8119233
sA      0.8560450 0.8560057 0.009332688 0.8377985 0.8739295
sB      0.7438864 0.7439627 0.014481510 0.7155535 0.7710848
```

Optimizing the Multinomial Likelihood - Movement Models - JAGS

Anonymized

Load packages and simulate the data

```
require (IPMbook)
require (jagsUI)
```

Let's simulate a dummy mark-recapture dataset to be used as an example, called y. Fix a seed first.

```
set.seed(2026)
```

Now simulate the data. Recall: movement model, two sites.

```
# Define data-generating values.

phiA <- 0.85 # Survival at site A
phiB <- 0.75 # Survival at site B
psiAB <- 0.2 # Movement from site A to site B
psiBA <- 0.2 # Movement from site B to site A
pA <- 0.7 # Detection probability at site A
pB <- 0.4 # Detection probability at site B

nyears <- 15 # Number of years
n.states <- 3 # Number of states (1: Alive at site A; 2: Alive at site B; 3:
Dead)
n.obs <- 3 # Number of observations (1: Seen Alive at A; 2: Seen Alive at B;
3: Not Seen)
nmarked <- 80 # Number of marked individuals per occasion/year

# Define vector of first encounters
f <- rep(1:(nyears-1), each=nmarked)
nind <- length(f) # Total number of individuals

# Define transition matrices.
# These are 4-dimensional matrices, with
# Dimension 1: state of departure
# Dimension 2: state of arrival
# Dimension 3: individual
# Dimension 4: time


# 1. State process matrix
```

```
PSI.STATE <- array(NA, dim = c(n.states, n.states, nind, nyears-1))

for(i in 1:nind){
  for(t in 1:(nyears-1)){

    PSI.STATE[,,i,t] <- matrix(c(

      phiA*(1-psiAB), phiA*psiAB, 1-phiA,
      phiB*psiBA, phiB*(1-psiBA), 1-phiB,
      0,0,1), nrow = n.states, byrow = TRUE)
  }
}

#2. Observation process matrix

PSI.OBS <- array(NA, dim = c(n.states, n.obs, nind, nyears-1))
for(i in 1:nind){
  for(t in 1:(nyears-1)){

    PSI.OBS[,,i,t] <- matrix(c(

      pA, 0, 1-pA,
      0, pB, 1-pB,
      0, 0, 1), nrow = n.states, byrow = TRUE

    )

  }
}

# State or ecological process
# Simulate true system state
z <- array(NA, dim=c(nind, nyears)) # Create empty true state matrix.
# Initial conditions: all individuals alive at f(i), but start either at site
 A or at B.
initial.state <- c(rep(1, round(nind/2)), rep(2, round(nind/2)))
for (i in 1:nind){
  z[i,f[i]] <- initial.state[i]
}

# Propagate true state process forwards via transition rule
for (i in 1:nind){
  for (t in (f[i]+1):nyears){
    departure.state <- z[i,t-1]
    arrival.state <- which(rmultinom(1,1, PSI.STATE[departure.state,,i,t-1])=
=1)
    z[i,t] <- arrival.state
  } #t
```

```
} #i

# Observation process: simulate observations using observation matrix
y <- array(3, dim=c(nind, nyears))
for (i in 1:nind){
  y[i,f[i]] <- z[i,f[i]]
  for (t in (f[i]+1):nyears){
    true.state <- z[i,t]
    observed.state <- which(rmultinom(1,1, PSI.OBS[true.state,,i,t-1])==1)
    y[i,t] <- observed.state
  } #t
} #i
```

Have a look at the first 6 rows of the mark-recapture dataset:

```
head (y)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,1
4]
[1,]    1    3    3    3    3    3    3    3    3     3     3     3     3
 3
[2,]    1    1    1    1    1    1    3    2    3     3     2     3     3
 3
[3,]    1    3    1    1    1    1    2    2    3     2     2     3     3
 3
[4,]    1    1    1    1    1    2    2    1    1     3     3     3     3
 3
[5,]    1    1    3    3    3    3    3    3    3     3     3     3     3
 3
[6,]    1    1    3    3    3    3    3    3    3     3     3     3     3
 3
     [,15]
[1,]     3
[2,]     2
[3,]     3
[4,]     3
[5,]     3
[6,]     3
```

The observation codes have the following meanings:

- 1 = detected alive at site A
- 2 = detected alive at site B
- 3 = not observed

Generate the conventional m-array

Here we transform the encounter histories into a conventional m-array. We need to define the 3 states of the model first:

- 1 = alive in site A
- 2 = alive in site B
- 3 = dead (this state is not observable)

Replace observation code “not observed” (3) with a 0. We do this because the function “marray” from package IPMbook needs the “non-observed” event to be a 0.

```
y[y == 3] <- 0
```

Set how many states are unobservable, and create the m-array using function “marray” from package IPMbook. Recall that there is one unobservable state: state 3 (dead), since in this model there are no dead recoveries.

```
unobs <- 1
marr <- marray(y, unobs = unobs)
```

Have a look at the m-array:

```
head (marr)
```

```
        recaptured
released Y2.S1 Y2.S2 Y2.S3 Y3.S1 Y3.S2 Y3.S3 Y4.S1 Y4.S2 Y4.S3 Y5.S1 Y5.S2
   Y1.S1    38     2     0    12     3     0     3     0     0     0     2
   Y1.S2     0     0     0     0     0     0     0     0     0     0     0
   Y1.S3     0     0     0     0     0     0     0     0     0     0     0
   Y2.S1     0     0     0    47     8     0    16     4     0     5     4
   Y2.S2     0     0     0     0     0     0     2     0     0     0     0
   Y2.S3     0     0     0     0     0     0     0     0     0     0     0
        recaptured
released Y5.S3 Y6.S1 Y6.S2 Y6.S3 Y7.S1 Y7.S2 Y7.S3 Y8.S1 Y8.S2 Y8.S3 Y9.S1
   Y1.S1     0     0     0     0     0     0     0     0     0     0     0
   Y1.S2     0     0     0     0     0     0     0     0     0     0     0
   Y1.S3     0     0     0     0     0     0     0     0     0     0     0
   Y2.S1     0     0     0     0     0     0     0     0     0     0     0
   Y2.S2     0     0     0     0     0     0     0     0     0     0     0
   Y2.S3     0     0     0     0     0     0     0     0     0     0     0
        recaptured
released Y9.S2 Y9.S3 Y10.S1 Y10.S2 Y10.S3 Y11.S1 Y11.S2 Y11.S3 Y12.S1 Y12.S2
   Y1.S1     0     0      0      0      0      0      0      0      0      0
   Y1.S2     0     0      0      0      0      0      0      0      0      0
   Y1.S3     0     0      0      0      0      0      0      0      0      0
   Y2.S1     0     0      0      0      0      0      0      0      0      0
   Y2.S2     0     0      0      0      0      0      0      0      0      0
   Y2.S3     0     0      0      0      0      0      0      0      0      0
        recaptured
```

```
released Y12.S3 Y13.S1 Y13.S2 Y13.S3 Y14.S1 Y14.S2 Y14.S3 Y15.S1 Y15.S2 Y15.S
3
   Y1.S1      0      0      0      0      0      0      0      0      0
0
   Y1.S2      0      0      0      0      0      0      0      0      0
0
   Y1.S3      0      0      0      0      0      0      0      0      0
0
   Y2.S1      0      0      0      0      0      0      0      0      0
0
   Y2.S2      0      0      0      0      0      0      0      0      0
0
   Y2.S3      0      0      0      0      0      0      0      0      0
0
        recaptured
released never
   Y1.S1    20
   Y1.S2     0
   Y1.S3     0
   Y2.S1    34
   Y2.S2     0
   Y2.S3     0
```

1. First part in the optimization process: simplify the m-array

Step 1A: Unobservable states

Here, we have the state 3 (dead) which is unobservable. We can therefore remove rows and columns belonging to state 3 as they do not provide information for inference.

```
marr.opti <- marr[!grepl("S3", rownames(marr)), !grepl("S3", colnames(marr))]
```

As opposite to age models, further simplification of the m-array is not possible.

Let's have a look at the resulting simplified m-array:

```
head (marr.opti)

        recaptured
released Y2.S1 Y2.S2 Y3.S1 Y3.S2 Y4.S1 Y4.S2 Y5.S1 Y5.S2 Y6.S1 Y6.S2 Y7.S1
   Y1.S1    38     2    12     3     3     0     0     2     0     0     0
   Y1.S2     0     0     0     0     0     0     0     0     0     0     0
   Y2.S1     0     0    47     8    16     4     5     4     0     0     0
   Y2.S2     0     0     0     0     2     0     0     0     0     0     0
   Y3.S1     0     0     0     0    64    12    15     8     4     3     3
   Y3.S2     0     0     0     0     0     2     0     1     0     1     0
        recaptured
released Y7.S2 Y8.S1 Y8.S2 Y9.S1 Y9.S2 Y10.S1 Y10.S2 Y11.S1 Y11.S2 Y12.S1
   Y1.S1     0     0     0     0     0      0      0      0      0      0
   Y1.S2     0     0     0     0     0      0      0      0      0      0
   Y2.S1     0     0     0     0     0      0      0      0      0      0
```

```
   Y2.S2     0     0     0     0     0     0     0     0     0     0
   Y3.S1     1     2     0     0     0     0     0     0     0     0
   Y3.S2     0     0     0     0     0     0     0     0     0     0
         recaptured
released Y12.S2 Y13.S1 Y13.S2 Y14.S1 Y14.S2 Y15.S1 Y15.S2 never
   Y1.S1      0      0      0      0      0      0      0    20
   Y1.S2      0      0      0      0      0      0      0     0
   Y2.S1      0      0      0      0      0      0      0    34
   Y2.S2      0      0      0      0      0      0      0     0
   Y3.S1      0      0      0      0      0      0      0    27
   Y3.S2      0      0      0      0      0      0      0     7
```

It is still smaller than the non-simplified m-array

```
compar_dim <- data.frame(row_number = c(nrow(marr), nrow(marr.opti)), col_num
ber = c(ncol(marr), ncol(marr.opti)))
rownames(compar_dim) = c("Original m-array", "Simplified m-array")

compar_dim
```

```
                  row_number col_number
Original m-array          42         43
Simplified m-array        28         29
```

Step 1B: Create vector of first encounters "year_first"

This is a vector giving the year of release for each row in the m-array. Formulating it prevents computation of state-year combinations before the state-year of release of the individual. It will be used in the calculation of π and in the multinomial likelihood.

```
row_names <- rownames(marr.opti)
year_first <- as.numeric(sub("Y([0-9]+)\\.S[0-9]+", "\\1", row_names))
```

It looks like this:

```
year_first
```

```
 [1]  1  1  2  2  3  3  4  4  5  5  6  6  7  7  8  8  9  9 10 10 11 11 12 12
13
[26] 13 14 14
```

Create a vector storing the site of first capture for each row of the m-array. This allows calculating the pathway of each cohort starting from their site of release.

```
site_of_capture <- as.numeric(sub("Y[0-9]+\\.S([0-9]+)", "\\1", row_names))
```

Which looks like:

```
site_of_capture
```

```
 [1] 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
```

Define the number of years:

```
nyears <- 15
```

Define the number of states in which individuals are released and re-encountered. Here it is 2 states in the two:

- Alive at site A
- Alive at site B

```
# Number of states of release
ns.rel <- 2

# Number of states of re-encounter
ns.obs <- 2
```

2. Second part in the optimization process: write the model

We will first show how to construct the model section by section, and next, we will provide the whole model.

Step 2A) Assemble the data

Pass a list with all necessary data to run the model.

```
my.data <- list(nyears = nyears,
                      ns.rel = ns.rel,
                      ns.obs = ns.obs,
                      year_first = year_first,
                      site_of_capture = site_of_capture,
                      marr = marr.opti,
                      rel = rowSums(marr.opti)
)
```

Note: rel is the number of individuals released in each row of the m-array, i.e., the sum of the rows of the m-array. It is also used in conventional multinomial models.

Step 2B) Create the model and define the priors

The first part of the model does not change: that where we define the priors of our parameters.

```
cat(file = "ms.jags", "


    model{

  # Define the priors

  # Survival
```

```
   sA ~ dbeta(1,1) # Vague prior for survival at site A
   sB ~ dbeta(1,1) # Vague prior for survival at site B

  # Movement: only between two sites

  psiA[1] ~ dbeta(1,1) # Vague prior for prob. of staying in site A
  psiA[2] <- 1-psiA[1] # Prob. of moving from site A to B is 1 - prob. of sta
ying in site A

  psiB[1] ~ dbeta(1,1) # Vague prior for prob. of moving from site B to A
  psiB[2] <- 1-psiB[1] # Prob. of staying in site B is 1 - prob. of moving fr
om site B to A

  # Detection probabilities

  pA ~ dbeta(1,1) # Vague prior for the detection probability in site A
  pB ~ dbeta(1,1) # Vague prior for the detection probability in site B
```

We additionally set all parameters in convenient structures for likelihood calculations.

First, we set all movement parameters into a matrix.

Second, we set all survival and detection parameters into a matrix with the number of columns equal to the number of years (years) minus one. This second step does not need to be done, as our parameters are time-constant, but we do it anyways so that it is easier to switch to a model with time-dependent parameters without needing to modify the likelihood.

```
# Define the movement matrix. Can be done with a loop if more sites

  mov [1,1] <- psiA[1] # probability of staying in A
  mov [1,2] <- psiA[2] # probability of moving from A to B
  mov [2,1] <- psiB[1] # probability of moving from B to A
  mov [2,2] <- psiB[2] # probability of staying in B

# Define survival as matrix (dependent on site and time, in case we want time
-dependence)

  for (t in 1:(nyears-1)) {

    s [1,t] <- sA # survival at A
    s [2,t] <- sB # survival at B

  }

 # Define detection probabilities as a matrix (dependent on site and time, in
 case we want time-dependence)
```

```
for (t in 1:(nyears-1)) {

  p [1,t] <- pA # detection at A
  p [2,t] <- pB # detection at B

}
```

Step 2C) Define π', U, and the optimized multinomial likelihood

Here we define the vector π', the equivalent to the vector π ("pi") in conventional multinomial models. Vector π' stores the probability for each cohort to be alive and detected in each site. The probabilities stored in π' are the cell probabilities used in the multinomial likelihood. Importantly, as opposite to the conventional implementation, in the optimized approach we calculate π' only from the possible pathways across state-year combinations of each cohort. This approach prevents calculation of structurally impossible transitions and results in an increase in computational efficiency.

As an auxiliary object to π' for pathway calculation purposes, we also define U. This object stores the probability of having survived and not having been observed before a subsequent observation.

2C1) Calculations for first year after release

Open the loop: "For each line in the new m-array, except the lines that indicate releases in the penultimate year":

```
for (t in 1:((nyears-2)*ns.rel)) {
```

First we perform all the calculations for the first year after release in the m-array.

The first year after release, the probability **U** to be alive and not detected in all sites is the product of the probabilities of:

1. Having survived in the site of release on the year of release.
2. Either have moved to another site or remain in the site of release
3. Not being detected the first year after release.

```
U [t,(year_first[t]*ns.obs-ns.obs+1):(year_first[t]*ns.obs)] <- s [site_of_ca
pture [t], year_first[t]] * mov[site_of_capture[t],1:ns.obs] * (1-p [1:ns.obs,
 year_first[t]])
```

The first year after release, the probability **π'** to be alive and detected in each site is the product of the probabilities of:

1. Having survived in the site of release on the year of release.
2. Either have moved to another site or remain in the site of release

3. Being detected the first year after release.

```
pi [t,(year_first[t]*ns.obs-ns.obs+1):(year_first[t]*ns.obs)] <- s [site_of_c
apture[t], year_first[t]] * mov[site_of_capture[t],1:ns.obs] * p [1:ns.obs, y
ear_first[t]]
```

2C2) Calculations for the following years except for releases in the penultimate year

Now we do the calculations for the following years after release in the m-array (up to the penultimate year). For this we open a nested loop.

```
for (n in (year_first[t]+1):(nyears-1)) {
```

We first calculate U for any given year t after the first year after release, up to the last year. Recall that here, for any given time after the first year after release, U is the product of the probabilities of:

1. Having survived between the year of release and t-1
2. Being in each site at t without having been detected at any year before t.
3. Surviving from t-1 to t in each site.
4. Moving from a site to another, or remaining at the original site, between t-1 and t.
5. Not being detected in each site at t.

Note that here the brackets are extremely important. You need to multiply U by survival before you multiply by the movement matrix. Otherwise, it will be considered that some individuals can move, then die.

```
 U [t,(n*ns.obs-ns.obs+1):(n*ns.obs)] <- (U [t,((n-1)*ns.obs-ns.obs+1):((n-1)
*ns.obs)] * s [1:ns.obs, n]) %*% mov[1:ns.obs,1:ns.obs] * (1-p [1:ns.obs, n])
```

We then calculate π', which for any given time t after the first year after release, is the product of the probabilities of:

1. Having survived between the year of release and t-1.
2. Being in each site at t without having been detected in any previous year.
3. Surviving from t-1 to t in each site.
4. Moving from a site to another, or remaining at the original site, between t-1 and t.
5. Being detected in each site at t.

Again pay attention to the brackets here for the same reason as stated before.

```
pi [t,(n*ns.obs-ns.obs+1):(n*ns.obs)] <- (U [t,((n-1)*ns.obs-ns.obs+1):((n-1)
*ns.obs)] * s [1:ns.obs, n]) %*% mov[1:ns.obs,1:ns.obs] * p [1:ns.obs, n]

  }
}
```

Note we just closed the brackets of the two loops.

2C3) Calculations for releases in the penultimate year

We now calculate π' for those lines that correspond to releases in the penultimate year in the m-array.

For these lines, we only need to calculate probabilities for the first year after release, which is the last year of the m-array.

Note we do not need to compute U for these lines, because we only need π' for the multinomial likelihood and U is only used to compute π' for the years after the first year after release.

Start by defining a loop for each line in the new m-array that corresponds to releases in the penultimate year. Follow the same logic as explained above for the first year after release.

```
for (t in ((nyears-2)*ns.rel+1):((nyears-1)*ns.rel)) {

pi [t,(year_first[t]*ns.obs-ns.obs+1):(year_first[t]*ns.obs)] <- s [site_of_c
apture[t], year_first[t]] * mov[site_of_capture[t],1:ns.obs] * p [1:ns.obs, y
ear_first[t]]

}
```

2C4) Calculate π' for never being re-encountered after release

The probability of being never re-encountered is 1 - the probability of being re-encountered in any of the sites and any of the years after release.

```
  for (t in 1:((nyears-1)*ns.rel)) {

    pi [t,1+(ns.obs*(nyears-1))] <- 1-sum (pi [t,(year_first[t]*ns.obs-(ns.ob
s-1)):(ns.obs*(nyears-1))])
```

2C5) Define the multinomial likelihood

We can do so within the same loop.

```
# Define multinomial likelihood

    marr [t,(year_first[t]*ns.obs-(ns.obs-1)):(nyears*ns.obs-(ns.obs-1))] ~ d
multi (pi [t,(year_first[t]*ns.obs-(ns.obs-1)):(nyears*ns.obs-(ns.obs-1))], r
```

```
el [t])

  }
```

And end the model.

```
}")
```

3. Third part in the optimization process: run the model

Here's the whole model.

```
cat(file = "ms.jags", "

    model{

  # Survival

  sA ~ dbeta(1,1)
  sB ~ dbeta(1,1)

  # Movement: only between two sites

  psiA[1] ~ dbeta(1,1)
  psiA[2] <- 1-psiA[1]
  psiB[1] ~ dbeta(1,1)
  psiB[2] <- 1-psiB[1]

  # Detection

  pA ~ dbeta(1,1)
  pB ~ dbeta(1,1)

  # Define the movement matrix

  mov [1,1] <- psiA[1]
  mov [1,2] <- psiA[2]
  mov [2,1] <- psiB[1]
  mov [2,2] <- psiB[2]

  # Set survival in a time- and site-dependent matrix

  for (t in 1:(nyears-1)) {

    s [1,t] <- sA
    s [2,t] <- sB

  }
```

```
  # Define detection probabilities in a time- and site-dependent matrix

  for (t in 1:(nyears-1)) {

    p [1,t] <- pA
    p [2,t] <- pB

  }

  # Define π' (named 'pi' in the code)
  # Calculate π' for first year after release

  for (t in 1:((nyears-2)*ns.rel)) {

    U [t,(year_first[t]*ns.obs-ns.obs+1):(year_first[t]*ns.obs)] <- s [site_o
f_capture [t], year_first[t]] * mov[site_of_capture[t],1:ns.obs] * (1-p [1:ns.
obs, year_first[t]])

    pi [t,(year_first[t]*ns.obs-ns.obs+1):(year_first[t]*ns.obs)] <- s [site_
of_capture[t], year_first[t]] *
      mov[site_of_capture[t],1:ns.obs] *
      p [1:ns.obs, year_first[t]]

    # Calculate π' for following years until penultimate year

    for (n in (year_first[t]+1):(nyears-1)) {

      U [t,(n*ns.obs-ns.obs+1):(n*ns.obs)] <- (U [t,((n-1)*ns.obs-ns.obs+1):
((n-1)*ns.obs)] * s [1:ns.obs, n]) %*% mov[1:ns.obs,1:ns.obs] * (1-p [1:ns.ob
s, n])

      pi [t,(n*ns.obs-ns.obs+1):(n*ns.obs)] <- (U [t,((n-1)*ns.obs-ns.obs+1):
((n-1)*ns.obs)] * s [1:ns.obs, n]) %*% mov[1:ns.obs,1:ns.obs] * p [1:ns.obs,
n]

    }
  }

    # Calculate π' for the penultimate year

  for (t in ((nyears-2)*ns.rel+1):((nyears-1)*ns.rel)) {

    pi [t,(year_first[t]*ns.obs-ns.obs+1):(year_first[t]*ns.obs)] <- s [site_
of_capture[t], year_first[t]] * mov[site_of_capture[t],1:ns.obs] * p [1:ns.ob
s, year_first[t]]

  }
```

```
  # Calculate π' for the penultimate year for individuals that were never re-
encountered

  for (t in 1:((nyears-1)*ns.rel)) {

    pi [t,1+(ns.obs*(nyears-1))] <- 1-sum (pi [t,(year_first[t]*ns.obs-(ns.ob
s-1)):(ns.obs*(nyears-1))])


    # Define multinomial likelihood

    marr [t,(year_first[t]*ns.obs-(ns.obs-1)):(nyears*ns.obs-(ns.obs-1))] ~ d
multi (pi [t,(year_first[t]*ns.obs-(ns.obs-1)):(nyears*ns.obs-(ns.obs-1))], r
el [t])

  }
}
")
```

Define initial values.

```
inits <- function(){list(
  sA = rbeta(1,1,1),
  sB = rbeta(1,1,1),
  pA = rbeta(1,1,1),
  pB = rbeta(1,1,1)
)}
```

Set parameters to save and MCMC settings

```
parameters = c("sA", "sB",  "pA", "pB", "psiA", "psiB")

ni <- 20000 # number of iterations
nb <- 1 # number of burn-in iterations
nc <- 4 # number of chains
nt <- 1 # thinning rate
```

Run the model

```
mcmc.out <- jags(my.data, inits, parameters, "ms.jags", n.iter = ni, n.burnin
 = nb, n.chains = nc, n.thin = nt, parallel = FALSE)


Processing function input.......

Done.

Compiling model graph
   Resolving undeclared variables
```

```
   Allocating nodes
Graph information:
   Observed stochastic nodes: 28
   Unobserved stochastic nodes: 6
   Total graph size: 1424

Initializing model

Adaptive phase.....
Adaptive phase complete


 Burn-in phase, 1 iterations x 4 chains


Sampling from joint posterior, 19999 iterations x 4 chains



Calculating statistics.......

Done.
```

And finally, show the summary for all chains.

```
print (mcmc.out)
```

```
JAGS output for model 'ms.jags', generated by jagsUI.
Estimates based on 4 chains of 20000 iterations,
adaptation = 100 iterations (sufficient),
burn-in = 1 iterations and thin rate = 1,
yielding 79996 total samples from the joint posterior.
MCMC ran for 1.079 minutes at time 2026-06-18 11:20:13.395651.

            mean    sd    2.5%     50%   97.5% overlap0 f Rhat n.eff
sA         0.856 0.009   0.838   0.856   0.874    FALSE 1    1 35243
sB         0.744 0.014   0.716   0.744   0.771    FALSE 1    1 79996
pA         0.679 0.016   0.647   0.679   0.711    FALSE 1    1 68505
pB         0.446 0.021   0.405   0.445   0.488    FALSE 1    1 46040
psiA[1]    0.809 0.013   0.783   0.809   0.833    FALSE 1    1 60708
psiA[2]    0.191 0.013   0.167   0.191   0.217    FALSE 1    1 60708
psiB[1]    0.215 0.014   0.188   0.215   0.244    FALSE 1    1 79996
psiB[2]    0.785 0.014   0.756   0.785   0.812    FALSE 1    1 79996
deviance 825.020 3.491 820.212 824.377 833.534    FALSE 1    1 26302

Successful convergence based on Rhat values (all < 1.1).
Rhat is the potential scale reduction factor (at convergence, Rhat=1).
For each parameter, n.eff is a crude measure of effective sample size.

overlap0 checks if 0 falls in the parameter's 95% credible interval.
```

```
f is the proportion of the posterior with the same sign as the mean;
i.e., our confidence that the parameter is positive or negative.

DIC info: (pD = var(deviance)/2)
pD = 6.1 and DIC = 831.113
DIC is an estimate of expected predictive error (lower is better).
```

Optimizing the Multinomial Likelihood - Age Models - NIMBLE

Anonymized

Load packages and simulate data

```
require (IPMbook)
require (nimble)
```

Let's simulate a dummy mark-recapture dataset to be used as an example, called y. Fix a seed first.

```
set.seed(2026)
```

Now simulate the data. Recall: age model, two age classes.

```
# Define the data-generating values

s1 <- .4 # Survival of age class 1 "juvenile"
s2 <- .8 # Survival of age class 2 "adult"
p1 <- .6 # Detection probability
r <- .2 # Dead recovery probability

nyears <- 15 # Number of years
n.states <- 4 # Number of states (1: Alive at age 1; 2: Alive at age 2, 3: Re
cently dead, 4: Long-dead)
n.obs <- 4 # Number of observations (1: Seen alive at age 1; 2: Seen alive at
 age 2; 3: Seen recently dead; 4: Not seen)
nmarked <- 80 # Number of released individuals per year/occasion

# Determine occasion when an individual first captured and marked
f <- rep(1:(nyears-1), each=nmarked)
nind <- length(f) # Total number of individuals

#States:
#1) Juveniles
#2) Adults
#3) Dead
#4) Long dead

# 1. State process matrix

    PSI.STATE <- matrix(c(

      0, s1, 1-s1, 0,
      0, s2, 1-s2, 0,
```

```
      0, 0, 0, 1,
      0, 0, 0, 1),
      nrow = n.states, byrow = TRUE)

#2. Observation process matrix

    PSI.OBS <- matrix(c(

      0, 0, 0, 1,
      0, p1, 0, 1-p1,
      0, 0, r, 1-r,
      0, 0, 0, 1), nrow = n.states, byrow = TRUE)


# State or ecological process
# Simulate true system state
z <- array(NA, dim=c(nind, nyears)) # Create an empty true state matrix
# Initial conditions: all individuals alive at age class 1 at f(i)
initial.state <- rep(1, length(f))
for (i in 1:nind){
  z[i,f[i]] <- initial.state[i]
}
# Propagate alive/dead process forwards via transition rule
for (i in 1:nind){
  for (t in (f[i]+1):nyears){
    departure.state <- z[i,t-1]
    arrival.state <- which(rmultinom(1,1, PSI.STATE[departure.state,])==1)
    z[i,t] <- arrival.state
  } #t
} #i

# Observation process: simulate observations using observation matrix
y <- array(max(z, na.rm = T), dim=c(nind, nyears))
for (i in 1:nind){
  y[i,f[i]] <- z[i,f[i]]
  for (t in (f[i]+1):nyears){
    true.state <- z[i,t]
    observed.state <- which(rmultinom(1,1, PSI.OBS[true.state,])==1)
    y[i,t] <- observed.state
  } #t
} #i
```

Have a look at the first 6 rows:

```
head (y)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,1
4]
[1,]    1    2    4    4    2    2    4    2    3     4     4     4     4
 4
```

```
[2,]    1    4    4    4    4    4    4    4    4    4    4    4    4
 4
[3,]    1    4    4    4    4    4    4    4    4    4    4    4    4
 4
[4,]    1    4    4    4    4    4    4    4    4    4    4    4    4
 4
[5,]    1    4    2    4    4    4    4    4    4    4    4    4    4
 4
[6,]    1    4    4    4    4    4    4    4    4    4    4    4    4
 4
     [,15]
[1,]     4
[2,]     4
[3,]     4
[4,]     4
[5,]     4
[6,]     4
```

The observation codes have the following meanings:

- 1 = observed alive at age class 1 (juvenile)
- 2 = observed alive at age class 2 (adult)
- 3 = found dead
- 4 = not observed

Generate the conventional m-array

Here we transform the encounter histories into a conventional m-array. We need to define the 4 states of the model first:

- 1 = alive as juvenile
- 2 = alive as adult
- 3 = recently dead (i.e., in the last year; can be found)
- 4 = long dead (i.e., more than one year ago; this state is not observable)

Replace observation code “not observed” (4) with a 0. We do this because the function “marray” from package IPMbook needs the “non-observed” event to be a 0.

```
y[y == 4] <- 0
```

Set how many states are unobservable, and create the m-array using function “marray” from package IPMbook. Recall: here there is a single unobservable state (state 4 “Long-dead”)

```
unobs <- 1
marr <- marray(y, unobs = unobs)
```

Have a look at the m-array:

```
head (marr)
```

```
        recaptured
released Y2.S1 Y2.S2 Y2.S3 Y2.S4 Y3.S1 Y3.S2 Y3.S3 Y3.S4 Y4.S1 Y4.S2 Y4.S3
   Y1.S1     0    16    10     0     0     3     0     0     0     1     0
   Y1.S2     0     0     0     0     0     0     0     0     0     0     0
   Y1.S3     0     0     0     0     0     0     0     0     0     0     0
   Y1.S4     0     0     0     0     0     0     0     0     0     0     0
   Y2.S1     0     0     0     0     0    18     8     0     0     8     0
   Y2.S2     0     0     0     0     0     7     0     0     0     3     0
        recaptured
released Y4.S4 Y5.S1 Y5.S2 Y5.S3 Y5.S4 Y6.S1 Y6.S2 Y6.S3 Y6.S4 Y7.S1 Y7.S2
   Y1.S1     0     0     0     0     0     0     0     0     0     0     0
   Y1.S2     0     0     0     0     0     0     0     0     0     0     0
   Y1.S3     0     0     0     0     0     0     0     0     0     0     0
   Y1.S4     0     0     0     0     0     0     0     0     0     0     0
   Y2.S1     0     0     3     0     0     0     0     0     0     0     0
   Y2.S2     0     0     2     0     0     0     0     0     0     0     0
        recaptured
released Y7.S3 Y7.S4 Y8.S1 Y8.S2 Y8.S3 Y8.S4 Y9.S1 Y9.S2 Y9.S3 Y9.S4 Y10.S1
   Y1.S1     0     0     0     0     0     0     0     0     0     0      0
   Y1.S2     0     0     0     0     0     0     0     0     0     0      0
   Y1.S3     0     0     0     0     0     0     0     0     0     0      0
   Y1.S4     0     0     0     0     0     0     0     0     0     0      0
   Y2.S1     0     0     0     0     0     0     0     1     0     0      0
   Y2.S2     0     0     0     0     0     0     0     0     0     0      0
        recaptured
released Y10.S2 Y10.S3 Y10.S4 Y11.S1 Y11.S2 Y11.S3 Y11.S4 Y12.S1 Y12.S2 Y12.S
3
   Y1.S1      0      0      0      0      0      0      0      0      0
0
   Y1.S2      0      0      0      0      0      0      0      0      0
0
   Y1.S3      0      0      0      0      0      0      0      0      0
0
   Y1.S4      0      0      0      0      0      0      0      0      0
0
   Y2.S1      0      0      0      0      0      0      0      0      0
0
   Y2.S2      0      0      0      0      0      0      0      0      0
0
        recaptured
released Y12.S4 Y13.S1 Y13.S2 Y13.S3 Y13.S4 Y14.S1 Y14.S2 Y14.S3 Y14.S4 Y15.S
1
   Y1.S1      0      0      0      0      0      0      0      0      0
0
   Y1.S2      0      0      0      0      0      0      0      0      0
0
```

```
   Y1.S3      0      0      0      0      0      0      0      0      0
0
   Y1.S4      0      0      0      0      0      0      0      0      0
0
   Y2.S1      0      0      0      0      0      0      0      0      0
0
   Y2.S2      0      0      0      0      0      0      0      0      0
0
         recaptured
released Y15.S2 Y15.S3 Y15.S4 never
   Y1.S1      0      0      0    50
   Y1.S2      0      0      0     0
   Y1.S3      0      0      0     0
   Y1.S4      0      0      0     0
   Y2.S1      0      0      0    42
   Y2.S2      0      0      0     4
```

1. First part in the optimization process: simplify the m-array

Step 1A: Unobservable states

Here, we have the state 4 (long dead) which is unobservable. We can therefore remove rows and columns belonging to state 4 as they do not provide information for inference.

```
marr.1 <- marr[!grepl("S4", rownames(marr)), !grepl("S4", colnames(marr))]
```

Step 1B: Impossible transitions

State 3 always transition to state 4 (long dead) in the next year. This is a transient observable state, so we can remove rows belonging to state 3.

```
marr.2 <- marr.1[!grepl("S3", rownames(marr.1)), ]
```

Step 1C: Summarising different age classes into one

States 1 and 2 are age-structured states; they differ only by the age class of the individuals (juvenile or adult). Columns that belong to these states can thus be combined by summing them

Note that in that case, it would have been possible as well to remove the columns belonging to state 1, as indeed, no re-encounter as juvenile is possible.

```
# Matrix with only columns of states 1 and 2
s12 <- marr.2[, grepl("S1$|S2$", colnames(marr.2)), drop = FALSE]

# Matrix with only columns of state 3
s3 <- marr.2[, grepl("S3$", colnames(marr.2)), drop = FALSE]

# "Never re-encountered" column
never <- marr.2[, "never", drop = FALSE]
```

```
# Combine columns of states 1 and 2 by summing them, and add columns of state
 3
marr.opti <- cbind(
  s12[, seq(1, ncol(s12), 2), drop = FALSE] + s12[, seq(2, ncol(s12), 2), dro
p = FALSE],
  s3,
  never
)

# Put columns in the correct order, with all states of each year together
marr.opti <- marr.opti[, c(as.vector(rbind(seq_len(ncol(s3)), ncol(s3) + seq_
len(ncol(s3)))), ncol(marr.opti)), drop = FALSE]
colnames(marr.opti) <- c(as.vector(rbind(
  paste0(sub("S1$", "", colnames(s12)[seq(1, ncol(s12), 2)]), "S1+S2"),
  colnames(s3)
)), "never")
```

Let's have a look at the resulting simplified m-array:

```
head (marr.opti)
```

```
      Y2.S1+S2 Y2.S3 Y3.S1+S2 Y3.S3 Y4.S1+S2 Y4.S3 Y5.S1+S2 Y5.S3 Y6.S1+S2
Y1.S1       16    10        3     0        1     0        0     0        0
Y1.S2        0     0        0     0        0     0        0     0        0
Y2.S1        0     0       18     8        8     0        3     0        0
Y2.S2        0     0        7     0        3     0        2     0        0
Y3.S1        0     0        0     0       23    11        6     0        2
Y3.S2        0     0        0     0       11     1        6     0        1
      Y6.S3 Y7.S1+S2 Y7.S3 Y8.S1+S2 Y8.S3 Y9.S1+S2 Y9.S3 Y10.S1+S2 Y10.S3
Y1.S1     0        0     0        0     0        0     0         0      0
Y1.S2     0        0     0        0     0        0     0         0      0
Y2.S1     0        0     0        0     0        1     0         0      0
Y2.S2     0        0     0        0     0        0     0         0      0
Y3.S1     0        1     0        0     0        0     0         0      0
Y3.S2     0        1     0        0     0        0     0         0      0
      Y11.S1+S2 Y11.S3 Y12.S1+S2 Y12.S3 Y13.S1+S2 Y13.S3 Y14.S1+S2 Y14.S3
Y1.S1         0      0         0      0         0      0         0      0
Y1.S2         0      0         0      0         0      0         0      0
Y2.S1         0      0         0      0         0      0         0      0
Y2.S2         0      0         0      0         0      0         0      0
Y3.S1         0      0         0      0         0      0         0      0
Y3.S2         0      0         0      0         0      0         0      0
      Y15.S1+S2 Y15.S3 never
Y1.S1         0      0    50
Y1.S2         0      0     0
Y2.S1         0      0    42
Y2.S2         0      0     4
Y3.S1         0      0    37
Y3.S2         0      0     8
```

It is substantially smaller than the non-simplified m-array

```
compar_dim <- data.frame(row_number = c(nrow(marr), nrow(marr.opti)), col_num
ber = c(ncol(marr), ncol(marr.opti)))
rownames(compar_dim) = c("Original m-array", "Simplified m-array")

compar_dim
```

```
                   row_number col_number
Original m-array           56         57
Simplified m-array         28         29
```

Step 1D: Create vector of first encounters "year_first"

This is a vector giving the year of release for each row in the m-array. Formulating it prevents computation of state-year combinations before the state-year of release of the individual. It will be used in the calculation of π and in the multinomial likelihood.

```
row_names <- rownames(marr.opti)
year_first <- as.numeric(sub("Y([0-9]+)\\.S[0-9]+", "\\1", row_names))
```

It looks like this:

```
year_first
```

```
 [1]  1  1  2  2  3  3  4  4  5  5  6  6  7  7  8  8  9  9 10 10 11 11 12 12
13
[26] 13 14 14
```

Define the number of years:

```
nyears <- 15
```

Define the number of states in which individuals are released and re-encountered. Here it is 2 states in the two.

States of release: alive as a juvenile and alive as an adult.

```
ns.rel <- 2
```

States of re-encounter: alive (both age classes combined) and dead (recently dead).

```
ns.obs <- 2
```

Step 1E: Create the age matrix

The age matrix stores the age class of individuals for each row of the m-array at each year from the year of release (year_first) to the penultimate year.

- We set as 0 the age class for the years before first encounter.
- We assign 1 to age class "juvenile"
- We assign 2 to age class "adult"

Formulating an age matrix allows a deterministic calculation of the pathways following age transitions, thus preventing us from considering impossible age transitions (i.e., those that have probability 0) in our calculations.

```
n_rows <- nrow(marr.opti)
age <- matrix(0, n_rows, nyears - 1)
for(i in 1:n_rows) {
  state_first <- as.numeric(sub(".*S([0-9]+)$", "\\1", row_names[i]))
  if(state_first == 1) {
    age[i, year_first[i]:(nyears - 1)] <- c(1, rep(2, nyears - year_first[i]
- 1))
  }
  if(state_first == 2) {
    age[i, year_first[i]:(nyears - 1)] <- 2
  }
}
```

Have a look at the age matrix we just created:

```
head(age)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,1
4]
[1,]    1    2    2    2    2    2    2    2    2     2     2     2     2
 2
[2,]    2    2    2    2    2    2    2    2    2     2     2     2     2
 2
[3,]    0    1    2    2    2    2    2    2    2     2     2     2     2
 2
[4,]    0    2    2    2    2    2    2    2    2     2     2     2     2
 2
[5,]    0    0    1    2    2    2    2    2    2     2     2     2     2
 2
[6,]    0    0    2    2    2    2    2    2    2     2     2     2     2
 2
```

2. Second part in the optimization process: write the model

We will first show how to construct the model section by section, and next, we will provide the whole model.

Step 2A) Assemble the data and the constants

Pass a list with all necessary data and constants to run the model.

```
# Constants
my.constants <- list(nyears = nyears,
                     ns.rel = ns.rel,
                     ns.obs = ns.obs,
                     year_first = year_first,
                     age = age)
```

```
# Data
my.data <- list(marr = marr.opti,
                rel = rowSums(marr.opti))
```

Note: rel is the number of individuals released in each row of the m-array, i.e., the sum of the rows of the m-array. It is also used in conventional multinomial models.

Step 2B) Create the model and define the priors

The first part of the model does not change: that where we define the priors of our parameters.

```
msopti <- nimbleCode({

  # Define the priors

    # Survival

  for(a in 1:2){

    s[a] ~ dbeta(1,1) # Vague prior for age-dependent survival probabilities

  }

  #Detection

  mean.p ~ dbeta(1,1) # Vague prior for the detection probability (alive)

  #Recovery
  mean.r ~ dbeta(1,1) # Vague prior for the dead-recovery probability
```

Step 2C) Define π' and the optimized multinomial likelihood

Here we define the vector π', the equivalent to the vector π ("pi") in conventional multinomial models. Vector π' stores the probability for each cohort to be alive and detected, or to be dead and recovered. The probabilities stored in π' are the cell probabilities used in the multinomial likelihood. Importantly, as opposite to the conventional implementation, in the optimized approach we calculate π' only from the possible pathways across state-year combinations of each cohort. This approach prevents calculation of structurally impossible transitions and results in an increase in computational efficiency.

As an auxiliary object to π' for pathway calculation purposes, we also define U. This object stores the probability of having survived and not having been observed before a subsequent observation.

2C1) Calculations for first year after release

Open the loop: “For each line in the new m-array, except the lines that indicate releases in the penultimate year”:

```
 for (t in 1:((nyears-2)*ns.rel)) {
```

First we perform all the calculations for the first year after release in the m-array.

The first year after release, the probability **U** to be alive and not detected is the probability to have survived (dependent on the age at release), multiplied by the probability not to be detected the first year after release.

```
U [t, year_first[t]] <- s [age [t, year_first [t]]] * (1-mean.p)
```

The first year after release, the probability **π’** to be alive and detected is the probability to have survived (dependent on the age at release), multiplied by the probability to be detected alive the first year after release.

```
 pi [t, year_first[t]*ns.obs-ns.obs+1] <- s [age [t, year_first [t]]] * mean.
p
```

The first year after release, the probability π’ to be dead and recovered is the probability to have died (dependent on the age at release), multiplied by the probability to be recovered dead the first year after release

```
 pi [t, year_first[t]*ns.obs] <- (1 - s [age [t, year_first [t]]]) * mean.r
```

2C2) Calculations for the following years except for releases in the penultimate year

Then we do the calculation for the following years after release in the m-array (up to the last year). For this we open a nested loop.

We first calculate U for any given year t after the first year after release. Recall that here, U [t, n-1] is the probability to have survived between release and t-1, multiplied by the age-dependent probability to survive from t-1 to t (s [age [t, n]]) multiplied by the probability not to be detected at t (1-mean.p).

Next we calculate π’.

- For any given year t after the first year after release, the probability π’ to be alive and detected is the probability U [t, n-1] to have survived between release and t-1, multiplied by the age-dependent probability to survive from t-1 to t (s [age [t, n]]) multiplied by the probability to be detected at t (mean.p).
- Likewise, for any given year t after the first year after release, the probability π’ to be dead and recovered is the probability U [t, n-1] to have survived between release and t-1, multiplied by the age-dependent probability to die from t-1 to t (1-s [age [t, n]]) multiplied by the probability to be recovered at t (mean.r)

And last, we close both loops.

```
 for (n in (year_first[t]+1):(nyears-1)) {

  U [t, n] <- U [t, n-1] * s [age [t, n]] * (1-mean.p)

  pi [t, n*ns.obs-ns.obs+1] <- U [t, n-1] * s [age [t, n]] *  mean.p

  pi [t, n*ns.obs] <- U [t, n-1] * (1 - s [age [t, n]]) * mean.r

    } # Close both loops
  }
```

2C3) Calculations for releases in the penultimate year

We now calculate π' for those lines that correspond to releases in the penultimate year in the m-array.

For these lines, we only need to calculate probabilities for the first year after release, which is the last year of the m-array.

Note we do not need to compute U for these lines, because we only need π' for the multinomial likelihood and U is only used to compute π' for the years after the first year after release.

Start by defining a loop for each line in the new m-array that corresponds to releases in the penultimate year. Follow the same logic as explained above for the first year after release.

```
for (t in ((nyears-2)*ns.rel+1):((nyears-1)*ns.rel)) {

  pi [t, year_first[t]*ns.obs-ns.obs+1] <- s [age [t, year_first [t]]] * mean.
p

  pi [t, year_first[t]*ns.obs] <- (1 - s [age [t, year_first [t]]]) * mean.r

}
```

2C4) Calculate π' for never being re-encountered after release

The probability of being never re-encountered is 1 - the probability of being re-encountered (both alive or dead) in any of the years after release.

```
  for (t in 1:((nyears-1)*ns.rel)) {

    pi [t, (nyears-1)*ns.obs + 1] <- 1 - sum(pi [t, (year_first[t]*ns.obs-(ns.
obs-1)):((nyears-1)*ns.obs)])
```

2C5) Define the multinomial likelihood

We can do so within the same loop.

```
# Define multinomial likelihood

    marr [t,(year_first[t]*ns.obs-(ns.obs-1)):(nyears*ns.obs-(ns.obs-1))] ~ d
multi (pi [t,(year_first[t]*ns.obs-(ns.obs-1)):(nyears*ns.obs-(ns.obs-1))], r
el [t])

  }
```

And end the model.

```
})
```

3. Third part in the optimization process: run the model

Here's the whole model.

```
msopti <- nimbleCode({

  #Define the priors

  # Age-structured Survival
  for(a in 1:2){

    s[a] ~ dbeta(1,1)

  }

  #Detection
  mean.p ~ dbeta(1,1)

  #Recovery
  mean.r ~ dbeta(1,1)

  # Define π' (named 'pi' in the code)
  # Calculate π' for first year after release

  for (t in 1:((nyears-2)*ns.rel)) {

    U [t, year_first[t]] <- s [age [t, year_first [t]]] *
      (1-mean.p)

    pi [t, year_first[t]*ns.obs-ns.obs+1] <- s [age [t, year_first [t]]] * me
an.p

    pi [t, year_first[t]*ns.obs] <- (1 - s [age [t, year_first [t]]]) * mean.
```

```
r

  # Calculate π' for following years until penultimate year

    for (n in (year_first[t]+1):(nyears-1)) {

      U [t, n] <- U [t, n-1] * s [age [t, n]] * (1-mean.p)

      pi [t, n*ns.obs-ns.obs+1] <- U [t, n-1] * s [age [t, n]] *
        mean.p

      pi [t, n*ns.obs] <- U [t, n-1] * (1 - s [age [t, n]]) *
        mean.r

    }
  }

  # Calculate π' for the penultimate year

  for (t in ((nyears-2)*ns.rel+1):((nyears-1)*ns.rel)) {

    pi [t, year_first[t]*ns.obs-ns.obs+1] <- s [age [t, year_first [t]]] * me
an.p

    pi [t, year_first[t]*ns.obs] <- (1 - s [age [t, year_first [t]]]) * mean.
r

  }

    # Calculate π' for the penultimate year for individuals that were never r
e-encountered

  for (t in 1:((nyears-1)*ns.rel)) {

    pi [t, (nyears-1)*ns.obs + 1] <- 1 - sum(pi [t, (year_first[t]*ns.obs-(ns.
obs-1)):((nyears-1)*ns.obs)])

    # Define multinomial likelihood

    marr [t,(year_first[t]*ns.obs-(ns.obs-1)):(nyears*ns.obs-(ns.obs-1))] ~ d
multi (pi [t,(year_first[t]*ns.obs-(ns.obs-1)):(nyears*ns.obs-(ns.obs-1))], r
el [t])

  }

})
```

Define initial values.

```
inits <- function(){list(
  s = rbeta(2,1,1),
  mean.p = rbeta(1,1,1),
  mean.r = rbeta(1,1,1)
)}
```

Set parameters to save and MCMC settings

```
parameters <- c("s", "mean.p", "mean.r")

ni <- 20000 # number of iterations
nb <- 1 # number of burn-in iterations
nc <- 4 # number of chains
nt <- 1 # thinning rate
```

Build, compile, and run the model

We could also do it with the single-line call of Nimble, but we will do it using the line-by-line call, which is equivalent.

```
# Build and compile the model
mod <- nimbleModel(code = msopti, constants = my.constants, data = my.data, i
nits = inits())

Defining model

Building model

Setting data and initial values

Running calculate on model
  [Note] Any error reports that follow may simply reflect missing values in m
odel variables.

Checking model sizes and dimensions

  [Note] This model is not fully initialized. This is not an error.
         To see which variables are not initialized, use model$initializeInfo
().
         For more information on model initialization, see help(modelInitiali
zation).

Cmod <- compileNimble(mod)

Compiling
  [Note] This may take a minute.
  [Note] Use 'showCompilerOutput = TRUE' to see C++ compilation details.

# Build and compile the MCMC
modelConf <- configureMCMC(mod, monitors = parameters)
```

```
===== Monitors =====
thin = 1: mean.p, mean.r, s
===== Samplers =====
RW sampler (4)
  - s[]  (2 elements)
  - mean.p
  - mean.r
```

```
modelMCMC <- buildMCMC(modelConf)
CmodelMCMC <- compileNimble(modelMCMC, project = mod)
```

```
Compiling
  [Note] This may take a minute.
  [Note] Use 'showCompilerOutput = TRUE' to see C++ compilation details.
```

```
# Run the MCMC
mcmc.out <- runMCMC(CmodelMCMC, niter = ni, nburnin = nb, thin = nt, nchains
= nc, progressBar = TRUE, samples = TRUE, summary = TRUE)
```

```
running chain 1...

|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|

running chain 2...

|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|

running chain 3...

|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|

running chain 4...

|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

And finally, show the summary for all chains.

```
print (mcmc.out$summary$all.chains)
```

```
           Mean    Median    St.Dev. 95%CI_low 95%CI_upp
mean.p 0.6139363 0.6141810 0.01441498 0.5857455 0.6417775
mean.r 0.1915927 0.1913376 0.01272637 0.1671464 0.2173031
s[1]   0.4129486 0.4128887 0.01683726 0.3806809 0.4462533
s[2]   0.8058755 0.8060085 0.01169238 0.7827203 0.8284667
```

Optimizing the Multinomial Likelihood - Age Models - JAGS

Anonymized

Load packages and data

```
require (IPMbook)
require (jagsUI)
```

Let's simulate a dummy mark-recapture dataset to be used as an example, called y. Fix a seed first.

```
set.seed(2026)
```

Now simulate the data. Recall: age model, two age classes.

```
# Define the data-generating values

s1 <- .4 # Survival of age class 1 "juvenile"
s2 <- .8 # Survival of age class 2 "adult"
p1 <- .6 # Detection probability
r <- .2 # Dead recovery probability

nyears <- 15 # Number of years
n.states <- 4 # Number of states (1: Alive at age 1; 2: Alive at age 2, 3: Re
cently dead, 4: Long-dead)
n.obs <- 4 # Number of observations (1: Seen alive at age 1; 2: Seen alive at
 age 2; 3: Seen recently dead; 4: Not seen)
nmarked <- 80 # Number of released individuals per year/occasion

# Determine occasion when an individual first captured and marked
f <- rep(1:(nyears-1), each=nmarked)
nind <- length(f) # Total number of individuals

#States:
#1) Juveniles
#2) Adults
#3) Dead
#4) Long dead

# 1. State process matrix

    PSI.STATE <- matrix(c(

      0, s1, 1-s1, 0,
      0, s2, 1-s2, 0,
```

```
      0, 0, 0, 1,
      0, 0, 0, 1),
      nrow = n.states, byrow = TRUE)

#2. Observation process matrix

    PSI.OBS <- matrix(c(

      0, 0, 0, 1,
      0, p1, 0, 1-p1,
      0, 0, r, 1-r,
      0, 0, 0, 1), nrow = n.states, byrow = TRUE)


# State or ecological process
# Simulate true system state
z <- array(NA, dim=c(nind, nyears)) # Create an empty true state matrix
# Initial conditions: all individuals alive at age class 1 at f(i)
initial.state <- rep(1, length(f))
for (i in 1:nind){
  z[i,f[i]] <- initial.state[i]
}
# Propagate alive/dead process forwards via transition rule
for (i in 1:nind){
  for (t in (f[i]+1):nyears){
    departure.state <- z[i,t-1]
    arrival.state <- which(rmultinom(1,1, PSI.STATE[departure.state,])==1)
    z[i,t] <- arrival.state
  } #t
} #i

# Observation process: simulate observations using observation matrix
y <- array(max(z, na.rm = T), dim=c(nind, nyears))
for (i in 1:nind){
  y[i,f[i]] <- z[i,f[i]]
  for (t in (f[i]+1):nyears){
    true.state <- z[i,t]
    observed.state <- which(rmultinom(1,1, PSI.OBS[true.state,])==1)
    y[i,t] <- observed.state
  } #t
} #i
```

Have a look at the first 6 rows:

```
head (y)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,1
4]
[1,]    1    2    4    4    2    2    4    2    3     4     4     4     4
 4
```

```
[2,]    1    4    4    4    4    4    4    4    4    4    4    4    4
 4
[3,]    1    4    4    4    4    4    4    4    4    4    4    4    4
 4
[4,]    1    4    4    4    4    4    4    4    4    4    4    4    4
 4
[5,]    1    4    2    4    4    4    4    4    4    4    4    4    4
 4
[6,]    1    4    4    4    4    4    4    4    4    4    4    4    4
 4
     [,15]
[1,]     4
[2,]     4
[3,]     4
[4,]     4
[5,]     4
[6,]     4
```

The observation codes have the following meanings:

- 1 = observed alive at age class 1 (juvenile)
- 2 = observed alive at age class 2 (adult)
- 3 = found dead
- 4 = not observed

Generate the conventional m-array

Here we transform the encounter histories into a conventional m-array. We need to define the 4 states of the model first:

- 1 = alive as juvenile
- 2 = alive as adult
- 3 = recently dead (i.e., in the last year; can be found)
- 4 = long dead (i.e., more than one year ago; this state is not observable)

Replace observation code “not observed” (4) with a 0. We do this because the function “marray” from package IPMbook needs the “non-observed” event to be a 0.

```
y[y == 4] <- 0
```

Set how many states are unobservable, and create the m-array using function “marray” from package IPMbook. Recall: here there is a single unobservable state (state 4 “Long-dead”)

```
unobs <- 1
marr <- marray(y, unobs = unobs)
```

Have a look at the m-array:

```
head (marr)
```

```
        recaptured
released Y2.S1 Y2.S2 Y2.S3 Y2.S4 Y3.S1 Y3.S2 Y3.S3 Y3.S4 Y4.S1 Y4.S2 Y4.S3
   Y1.S1     0    16    10     0     0     3     0     0     0     1     0
   Y1.S2     0     0     0     0     0     0     0     0     0     0     0
   Y1.S3     0     0     0     0     0     0     0     0     0     0     0
   Y1.S4     0     0     0     0     0     0     0     0     0     0     0
   Y2.S1     0     0     0     0     0    18     8     0     0     8     0
   Y2.S2     0     0     0     0     0     7     0     0     0     3     0
        recaptured
released Y4.S4 Y5.S1 Y5.S2 Y5.S3 Y5.S4 Y6.S1 Y6.S2 Y6.S3 Y6.S4 Y7.S1 Y7.S2
   Y1.S1     0     0     0     0     0     0     0     0     0     0     0
   Y1.S2     0     0     0     0     0     0     0     0     0     0     0
   Y1.S3     0     0     0     0     0     0     0     0     0     0     0
   Y1.S4     0     0     0     0     0     0     0     0     0     0     0
   Y2.S1     0     0     3     0     0     0     0     0     0     0     0
   Y2.S2     0     0     2     0     0     0     0     0     0     0     0
        recaptured
released Y7.S3 Y7.S4 Y8.S1 Y8.S2 Y8.S3 Y8.S4 Y9.S1 Y9.S2 Y9.S3 Y9.S4 Y10.S1
   Y1.S1     0     0     0     0     0     0     0     0     0     0      0
   Y1.S2     0     0     0     0     0     0     0     0     0     0      0
   Y1.S3     0     0     0     0     0     0     0     0     0     0      0
   Y1.S4     0     0     0     0     0     0     0     0     0     0      0
   Y2.S1     0     0     0     0     0     0     0     1     0     0      0
   Y2.S2     0     0     0     0     0     0     0     0     0     0      0
        recaptured
released Y10.S2 Y10.S3 Y10.S4 Y11.S1 Y11.S2 Y11.S3 Y11.S4 Y12.S1 Y12.S2 Y12.S
3
   Y1.S1      0      0      0      0      0      0      0      0      0
0
   Y1.S2      0      0      0      0      0      0      0      0      0
0
   Y1.S3      0      0      0      0      0      0      0      0      0
0
   Y1.S4      0      0      0      0      0      0      0      0      0
0
   Y2.S1      0      0      0      0      0      0      0      0      0
0
   Y2.S2      0      0      0      0      0      0      0      0      0
0
        recaptured
released Y12.S4 Y13.S1 Y13.S2 Y13.S3 Y13.S4 Y14.S1 Y14.S2 Y14.S3 Y14.S4 Y15.S
1
   Y1.S1      0      0      0      0      0      0      0      0      0
0
   Y1.S2      0      0      0      0      0      0      0      0      0
0
```

```
   Y1.S3      0      0      0      0      0      0      0      0      0
0
   Y1.S4      0      0      0      0      0      0      0      0      0
0
   Y2.S1      0      0      0      0      0      0      0      0      0
0
   Y2.S2      0      0      0      0      0      0      0      0      0
0
         recaptured
released Y15.S2 Y15.S3 Y15.S4 never
   Y1.S1      0      0      0    50
   Y1.S2      0      0      0     0
   Y1.S3      0      0      0     0
   Y1.S4      0      0      0     0
   Y2.S1      0      0      0    42
   Y2.S2      0      0      0     4
```

1. First part in the optimization process: simplify the m-array

Step 1A: Unobservable states

Here, we have the state 4 (long dead) which is unobservable. We can therefore remove rows and columns belonging to state 4 as they do not provide information for inference.

```
marr.1 <- marr[!grepl("S4", rownames(marr)), !grepl("S4", colnames(marr))]
```

Step 1B: Impossible transitions

State 3 always transition to state 4 (long dead) in the next year. This is a transient observable state, so we can remove rows belonging to state 3.

```
marr.2 <- marr.1[!grepl("S3", rownames(marr.1)), ]
```

Step 1C: Summarising different age classes into one

States 1 and 2 are age-structured states; they differ only by the age class of the individuals (juvenile or adult). Columns that belong to these states can thus be combined by summing them

Note that in that case, it would have been possible as well to remove the columns belonging to state 1, as indeed, no re-encounter as juvenile is possible.

```
# Matrix with only columns of states 1 and 2
s12 <- marr.2[, grepl("S1$|S2$", colnames(marr.2)), drop = FALSE]

# Matrix with only columns of state 3
s3 <- marr.2[, grepl("S3$", colnames(marr.2)), drop = FALSE]

# "Never re-encountered" column
never <- marr.2[, "never", drop = FALSE]
```

```
# Combine columns of states 1 and 2 by summing them, and add columns of state
 3
marr.opti <- cbind(
  s12[, seq(1, ncol(s12), 2), drop = FALSE] + s12[, seq(2, ncol(s12), 2), dro
p = FALSE],
  s3,
  never
)

# Put columns in the correct order, with all states of each year together
marr.opti <- marr.opti[, c(as.vector(rbind(seq_len(ncol(s3)), ncol(s3) + seq_
len(ncol(s3)))), ncol(marr.opti)), drop = FALSE]
colnames(marr.opti) <- c(as.vector(rbind(
  paste0(sub("S1$", "", colnames(s12)[seq(1, ncol(s12), 2)]), "S1+S2"),
  colnames(s3)
)), "never")
```

Let's have a look at the resulting simplified m-array:

```
head (marr.opti)
```

```
      Y2.S1+S2 Y2.S3 Y3.S1+S2 Y3.S3 Y4.S1+S2 Y4.S3 Y5.S1+S2 Y5.S3 Y6.S1+S2
Y1.S1       16    10        3     0        1     0        0     0        0
Y1.S2        0     0        0     0        0     0        0     0        0
Y2.S1        0     0       18     8        8     0        3     0        0
Y2.S2        0     0        7     0        3     0        2     0        0
Y3.S1        0     0        0     0       23    11        6     0        2
Y3.S2        0     0        0     0       11     1        6     0        1
      Y6.S3 Y7.S1+S2 Y7.S3 Y8.S1+S2 Y8.S3 Y9.S1+S2 Y9.S3 Y10.S1+S2 Y10.S3
Y1.S1     0        0     0        0     0        0     0         0      0
Y1.S2     0        0     0        0     0        0     0         0      0
Y2.S1     0        0     0        0     0        1     0         0      0
Y2.S2     0        0     0        0     0        0     0         0      0
Y3.S1     0        1     0        0     0        0     0         0      0
Y3.S2     0        1     0        0     0        0     0         0      0
      Y11.S1+S2 Y11.S3 Y12.S1+S2 Y12.S3 Y13.S1+S2 Y13.S3 Y14.S1+S2 Y14.S3
Y1.S1         0      0         0      0         0      0         0      0
Y1.S2         0      0         0      0         0      0         0      0
Y2.S1         0      0         0      0         0      0         0      0
Y2.S2         0      0         0      0         0      0         0      0
Y3.S1         0      0         0      0         0      0         0      0
Y3.S2         0      0         0      0         0      0         0      0
      Y15.S1+S2 Y15.S3 never
Y1.S1         0      0    50
Y1.S2         0      0     0
Y2.S1         0      0    42
Y2.S2         0      0     4
Y3.S1         0      0    37
Y3.S2         0      0     8
```

It is substantially smaller than the non-simplified m-array

```
compar_dim <- data.frame(row_number = c(nrow(marr), nrow(marr.opti)), col_num
ber = c(ncol(marr), ncol(marr.opti)))
rownames(compar_dim) = c("Original m-array", "Simplified m-array")

compar_dim
```

```
                   row_number col_number
Original m-array           56         57
Simplified m-array         28         29
```

Step 1D: Create vector of first encounters "year_first"

This is a vector giving the year of release for each row in the m-array. Formulating it prevents computation of state-year combinations before the state-year of release of the individual. It will be used in the calculation of π and in the multinomial likelihood.

```
row_names <- rownames(marr.opti)
year_first <- as.numeric(sub("Y([0-9]+)\\.S[0-9]+", "\\1", row_names))
```

It looks like this:

```
year_first
```

```
 [1]  1  1  2  2  3  3  4  4  5  5  6  6  7  7  8  8  9  9 10 10 11 11 12 12
13
[26] 13 14 14
```

Define the number of years:

```
nyears <- 15
```

Define the number of states in which individuals are released and re-encountered. Here it is 2 states in the two.

States of release: alive as a juvenile and alive as an adult.

```
ns.rel <- 2
```

States of re-encounter: alive (both age classes combined) and dead (recently dead).

```
ns.obs <- 2
```

Step 1E: Create the age matrix

The age matrix stores the age class of individuals for each row of the m-array at each year from the year of release (year_first) to the penultimate year.

- We set as 0 the age class for the years before first encounter.
- We assign 1 to age class "juvenile"
- We assign 2 to age class "adult"

Formulating an age matrix allows a deterministic calculation of the pathways following age transitions, thus preventing us from considering impossible age transitions (i.e., those that have probability 0) in our calculations.

```
n_rows <- nrow(marr.opti)
age <- matrix(0, n_rows, nyears - 1)
for(i in 1:n_rows) {
  state_first <- as.numeric(sub(".*S([0-9]+)$", "\\1", row_names[i]))
  if(state_first == 1) {
    age[i, year_first[i]:(nyears - 1)] <- c(1, rep(2, nyears - year_first[i]
- 1))
  }
  if(state_first == 2) {
    age[i, year_first[i]:(nyears - 1)] <- 2
  }
}
```

Have a look at the age matrix we just created:

```
head(age)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,1
4]
[1,]    1    2    2    2    2    2    2    2    2     2     2     2     2
 2
[2,]    2    2    2    2    2    2    2    2    2     2     2     2     2
 2
[3,]    0    1    2    2    2    2    2    2    2     2     2     2     2
 2
[4,]    0    2    2    2    2    2    2    2    2     2     2     2     2
 2
[5,]    0    0    1    2    2    2    2    2    2     2     2     2     2
 2
[6,]    0    0    2    2    2    2    2    2    2     2     2     2     2
 2
```

2. Second part in the optimization process: write the model

We will first show how to construct the model section by section, and next, we will provide the whole model.

Step 2A) Assemble the data

Pass a list with all necessary data to run the model.

```
my.data <- list(nyears = nyears,
                      ns.rel = ns.rel,
                      ns.obs = ns.obs,
                      year_first = year_first,
                      age = age,
                      marr = marr.opti,
```

```
                    rel = rowSums(marr.opti)
)
```

Note: rel is the number of individuals released in each row of the m-array, i.e., the sum of the rows of the m-array. It is also used in conventional multinomial models.

Step 2B) Create the model and define the priors

The first part of the model does not change: that where we define the priors of our parameters.

```
cat(file = "ms.jags", "
    model{

  # Define the priors

  # Survival

  for(a in 1:2){

    s[a] ~ dbeta(1,1) # Vague prior for age-dependent survival probabilities

  }

  #Detection

  mean.p ~ dbeta(1,1) # Vague prior for the detection probability (alive)

  #Recovery

  mean.r ~ dbeta(1,1) # Vague prior for the dead-recovery probability
```

Step 2C) Define π' and the optimized multinomial likelihood

Here we define the vector π', the equivalent to the vector π ("pi") in conventional multinomial models. Vector π' stores the probability for each cohort to be alive and detected, or to be dead and recovered. The probabilities stored in π' are the cell probabilities used in the multinomial likelihood. Importantly, as opposite to the conventional implementation, in the optimized approach we calculate π' only from the possible pathways across state-year combinations of each cohort. This approach prevents calculation of structurally impossible transitions and results in an increase in computational efficiency.

As an auxiliary object to π' for pathway calculation purposes, we also define U. This object stores the probability of having survived and not having been observed before a subsequent observation.

2C1) Calculations for first year after release

Open the loop: “For each line in the new m-array, except the lines that indicate releases in the penultimate year”:

```
for (t in 1:((nyears-2)*ns.rel)) {
```

First we perform all the calculations for the first year after release in the m-array.

The first year after release, the probability **U** to be alive and not detected is the probability to have survived (dependent on the age at release), multiplied by the probability not to be detected the first year after release.

```
U [t, year_first[t]] <- s [age [t, year_first [t]]] * (1-mean.p)
```

The first year after release, the probability **π’** to be alive and detected is the probability to have survived (dependent on the age at release), multiplied by the probability to be detected alive the first year after release.

```
pi [t, year_first[t]*ns.obs-ns.obs+1] <- s [age [t, year_first [t]]] * mean.p
```

The first year after release, the probability π’ to be dead and recovered is the probability to have died (dependent on the age at release), multiplied by the probability to be recovered dead the first year after release

```
pi [t, year_first[t]*ns.obs] <- (1 - s [age [t, year_first [t]]]) * mean.r
```

2C2) Calculations for the following years except for releases in the penultimate year

Then we do the calculation for the following years after release in the m-array (up to the last year). For this we open a nested loop.

We first calculate U for any given year t after the first year after release. Recall that here, U [t, n-1] is the probability to have survived between release and t-1, multiplied by the age-dependent probability to survive from t-1 to t (s [age [t, n]]) multiplied by the probability not to be detected at t (1-mean.p).

Next we calculate π’.

- For any given year t after the first year after release, the probability π’ to be alive and detected is the probability U [t, n-1] to have survived between release and t-1, multiplied by the age-dependent probability to survive from t-1 to t (s [age [t, n]]) multiplied by the probability to be detected at t (mean.p).

- Likewise, for any given year t after the first year after release, the probability π’ to be dead and recovered is the probability U [t, n-1] to have survived between release and t-1, multiplied by the age-dependent probability to die from t-1 to t (1-s [age [t, n]]) multiplied by the probability to be recovered at t (mean.r)

And last, we close both loops.

```
for (n in (year_first[t]+1):(nyears-1)) {

  U [t, n] <- U [t, n-1] * s [age [t, n]] *(1-mean.p)

  pi [t, n*ns.obs-ns.obs+1] <- U [t, n-1] * s [age [t, n]] * mean.p

  pi [t, n*ns.obs] <- U [t, n-1] * (1 - s [age [t, n]]) * mean.r

    } # Close both loops
  }
```

2C3) Calculations for releases in the penultimate year

We now calculate π' for those lines that correspond to releases in the penultimate year in the m-array.

For these lines, we only need to calculate probabilities for the first year after release, which is the last year of the m-array.

Note we do not need to compute U for these lines, because we only need π' for the multinomial likelihood and U is only used to compute π' for the years after the first year after release.

Start by defining a loop for each line in the new m-array that corresponds to releases in the penultimate year. Follow the same logic as explained above for the first year after release.

```
for (t in ((nyears-2)*ns.rel+1):((nyears-1)*ns.rel)) {

  pi [t, year_first[t]*ns.obs-ns.obs+1] <- s [age [t, year_first [t]]] * mean.
p

  pi [t, year_first[t]*ns.obs] <- (1 - s [age [t, year_first [t]]]) * mean.r

}
```

2C4) Calculate π' for never being re-encountered after release

The probability of being never re-encountered is 1 - the probability of being re-encountered (both alive or dead) in any of the years after release.

```
  for (t in 1:((nyears-1)*ns.rel)) {

    pi [t, (nyears-1)*ns.obs + 1] <- 1 - sum(pi [t, (year_first[t]*ns.obs-(ns.
obs-1)):((nyears-1)*ns.obs)])
```

2C5) Define the multinomial likelihood

We can do so within the same loop.

```
# Define multinomial likelihood

    marr [t,(year_first[t]*ns.obs-(ns.obs-1)):(nyears*ns.obs-(ns.obs-1))] ~ d
multi (pi [t,(year_first[t]*ns.obs-(ns.obs-1)):(nyears*ns.obs-(ns.obs-1))], r
el [t])

  }
```

And end the model.

```
}")
```

3. Third part in the optimization process: run the model

Here's the whole model.

```
cat(file = "ms.jags", "

    model{

  #Priors for age-structured survival

  for(a in 1:2){

    s[a] ~ dbeta(1,1)

  }

  #Detection

  mean.p ~ dbeta(1,1)

  #Recovery

  mean.r ~ dbeta(1,1)

  # Define π' (named 'pi' in the code)
  # Calculate π' for first year after release

  for (t in 1:((nyears-2)*ns.rel)) {

    U [t, year_first[t]] <- s [age [t, year_first [t]]] * (1-mean.p)

    pi [t, year_first[t]*ns.obs-ns.obs+1] <- s [age [t, year_first [t]]] * me
an.p

    pi [t, year_first[t]*ns.obs] <- (1 - s [age [t, year_first [t]]]) * mean.
r
```

```
  # Calculate π' for following years until penultimate year

    for (n in (year_first[t]+1):(nyears-1)) {

      U [t, n] <- U [t, n-1] * s [age [t, n]] * (1-mean.p)

      pi [t, n*ns.obs-ns.obs+1] <- U [t, n-1] * s [age [t, n]] *  mean.p

      pi [t, n*ns.obs] <- U [t, n-1] * (1 - s [age [t, n]]) * mean.r

    }
  }

  # Calculate π' for the penultimate year

  for (t in ((nyears-2)*ns.rel+1):((nyears-1)*ns.rel)) {

    pi [t, year_first[t]*ns.obs-ns.obs+1] <- s [age [t, year_first [t]]] * me
an.p

    pi [t, year_first[t]*ns.obs] <- (1 - s [age [t, year_first [t]]]) * mean.
r

   }

    # Calculate π' for the penultimate year for individuals that were never r
e-encountered

  for (t in 1:((nyears-1)*ns.rel)) {

    pi [t, (nyears-1)*ns.obs + 1] <- 1 - sum(pi [t, (year_first[t]*ns.obs-(ns.
obs-1)):((nyears-1)*ns.obs)])

    # Define multinomial likelihood

    marr [t,(year_first[t]*ns.obs-(ns.obs-1)):(nyears*ns.obs-(ns.obs-1))] ~ d
multi (pi [t,(year_first[t]*ns.obs-(ns.obs-1)):(nyears*ns.obs-(ns.obs-1))], r
el [t])

  }

}")
```

Define initial values.

```
inits <- function(){list(
  s = rbeta(2,1,1),
```

```
  mean.p = rbeta(1,1,1),
  mean.r = rbeta(1,1,1)
)}
```

Set parameters to save and MCMC settings

```
parameters <- c("s", "mean.p", "mean.r")

ni <- 20000 # number of iterations
nb <- 1 # number of burn-in iterations
nc <- 4 # number of chains
nt <- 1 # thinning rate
```

Run the model

```
mcmc.out <- jags(my.data, inits, parameters, "ms.jags", n.iter = ni, n.burnin
 = nb, n.chains = nc, n.thin = nt, parallel = FALSE)


Processing function input.......

Done.

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 28
   Unobserved stochastic nodes: 4
   Total graph size: 1047

Initializing model

Adaptive phase.....
Adaptive phase complete


 Burn-in phase, 1 iterations x 4 chains


Sampling from joint posterior, 19999 iterations x 4 chains


Calculating statistics.......

Done.
```

And finally, show the summary for all chains.

```
print (mcmc.out)
```

```
JAGS output for model 'ms.jags', generated by jagsUI.
Estimates based on 4 chains of 20000 iterations,
adaptation = 100 iterations (sufficient),
burn-in = 1 iterations and thin rate = 1,
yielding 79996 total samples from the joint posterior.
MCMC ran for 0.072 minutes at time 2026-06-18 11:18:12.235428.

            mean    sd    2.5%     50%   97.5% overlap0 f Rhat n.eff
s[1]       0.413 0.016   0.381   0.413   0.445    FALSE 1    1 46034
s[2]       0.806 0.011   0.783   0.806   0.828    FALSE 1    1 68103
mean.p     0.614 0.014   0.586   0.614   0.642    FALSE 1    1 79996
mean.r     0.192 0.013   0.167   0.191   0.217    FALSE 1    1 61089
deviance 616.328 2.798 612.840 615.694 623.379    FALSE 1    1 36396

Successful convergence based on Rhat values (all < 1.1).
Rhat is the potential scale reduction factor (at convergence, Rhat=1).
For each parameter, n.eff is a crude measure of effective sample size.

overlap0 checks if 0 falls in the parameter's 95% credible interval.
f is the proportion of the posterior with the same sign as the mean;
i.e., our confidence that the parameter is positive or negative.

DIC info: (pD = var(deviance)/2)
pD = 3.9 and DIC = 620.244
DIC is an estimate of expected predictive error (lower is better).
```